R. Joseph Trojan, CA Bar No. 137,067
Trojan@trojanlawoffices.com
TROJAN LAW OFFICES
9250 Wilshire Blvd., Suite 325
Beverly Hills, CA 90212
Telephone: 310-777-8399
Facsimile: 310-777-8348
Attorneys for Plaintiff,
PHOENIX SOLUTIONS, INC.

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

PHOENIX SOLUTIONS, INC., a California corporation, Plaintiff, v. WELLS FARGO BANK, N.A., a Delaware corporation, Defendant.	CASE NO. CV08-00863 MHP **DECLARATION OF R. JOSEPH TROJAN IN SUPPORT OF PHOENIX'S BRIEF REGARDING THE SCOPE OF WAIVER** Judge: Hon. Marilyn Hall Patel Trial Date: TBD Hearing Date: September 11, 2008 at 3:00 PM

I, R. Joseph Trojan, declare as follows:

1. I am an attorney at law, duly licensed to practice law in the State of California and the United States District Court for the Northern District of California. I am the owner of the Trojan Law Offices, the attorneys of record for Plaintiff, Phoenix Solutions, Inc. ("Phoenix" or "Plaintiff"). I have personal knowledge of the facts stated herein. If called upon to do so, I could and would completely testify that:

2. Plaintiff has previously produced communications between Dr. Ian Bennett and his patent counsel, Mr. J. Nicholas Gross. The documents are Bates numbered PHO006210 to PHO006351. These documents relate to the drafting and editing of the written description portion of the patents-in-suit and predate the applications' filing date. Attached hereto as Exhibit 1 is a true and correct copy of these documents, Bates numbered PHO006210 to PHO006351.

3. In a letter dated August 28, 2008, Defendant listed the following subject matter areas it argues that Plaintiff has waived privilege: (1) editing and drafting of the specification of the patents, including the scope and content of the prior art; (2) inventorship of the patents; (3)

novelty and/or innovation of the patents; (4) how to uncover prior art; (5) how prior art applies to the invention; and (6) how to draft the claims to avoid prior art. Attached hereto as Exhibit 2 is a true and correct copy of Defendant's August 28, 2008 letter.

4. The parties were unable to reach a consensus on the exact scope of the waiver. Although Defendant's August 28, 2008 letter claims that the parties reached some agreement regarding the scope of waiver, the letter left out language critical to the agreement. Plaintiff addresses this issue in a letter dated September 3, 2008. Attached hereto as Exhibit 3 is a true and correct copy of Plaintiff's September 3, 2008 letter.

I declare under penalty of perjury under the laws of the United States that the foregoing is true and correct.

Executed this 8th day of September, 2008 in Beverly Hills, California.

Respectfully submitted,

TROJAN LAW OFFICES
by

/s/ R. Joseph Trojan

R. Joseph Trojan
Attorney for Plaintiff,
PHOENIX SOLUTIONS, INC.

Subj: **Draft of Patent**
Date: 8/14/99 10:09:14 AM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: nicdagreek@aol.com

Nick:

I just wanted to remind you that we are working on the patent documentation that I intend to send to you by tomorrow Sunday. I know you are leaving on the 16th, and hope you will be checking your mail the day before your trip.

Regards, Ian

Ian Bennett
Principal
Phoenix Solutions, Inc.
634 Georgia Ave., Suite 101
Palo Alto, CA 94306, U.S.A.
Tel: (650) 858-0585; Fax: (650) 858-0586
E-mail: imb@best.com
URL: http://www.cyberscholars.com

----------------------- Headers --------------------------------
Return-Path:
Received: from aol.com (rly-zd05.mail.aol.com [172.31.33.229]) by air-zd01.mail.aol.com (v60.25) with ESMTP; Sat, 14 Aug 1999 13:09:14 -0400
Received: from proxy3.ba.best.com (proxy3.ba.best.com [206.184.139.14]) by rly-zd05.mx.aol.com (v60.25) with ESMTP; Sat, 14 Aug 1999 13:09:03 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy3.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id KAA02216
for ; Sat, 14 Aug 1999 10:08:50 -0700 (PDT)
Message-Id: <4.2.0.58.19990814100512.00a2dd00@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com (Unverified)
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Sat, 14 Aug 1999 10:10:39 -0700
To: nicdagreek@aol.com
From: Ian Bennett
Subject: Draft of Patent
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"; format=flowed

SUBJECT: SPEECH RECOGNITION OVER THE INTERNET

DATE: APRIL 28, 1999

There are basically three ways to implement speech recognition over the Internet:

1. To have the entire speech recognition engine on the user's machine – client, and implement all the functions of SR – pre-processing, MFCC extraction, and decoding on this client .
2. To have the client encode the raw speech and transmit the speech signals in digital form to the server where the speech decoding is done.
3. To have the speech recognition front-end at the client extract the MFCCs and then send these MFCCs to the server where final speech decoding is done using acoustic models, grammars and dictionaries.

When we first faced the issue of choosing between the above three alternatives, we assumed that we wanted to do the main speech recognition functions on the server for the following reasons:

1. To achieve a consistent and high recognition accuracy or lowest word error rate. To do this we assumed that we could apply a powerful server with known and controllable parameters to the SR problem. The configuration of such a server would be as follows: 800 MHz Pentium III or better (2 or 4 CPUs depending on cost at the time), 1 GB RAM and highest speed local bus. If we compare this to the typical "thin" client running at 200 MHz, 128 MB memory and lower speed bus, then it is obvious that the server configuration should give better performance. The average user however may have a much "thinner" client. And another point is that at any time, the server technology that we can apply will be much better suited, i.e more powerful for speech recognition than the user's client.
2. Control of the SR Consistency of Performance – if we put the SR on the client, we would not be able to control the large variation of the types of clients that are out there attached to the Internet. There are PCs with various processors ranging from Pentium I/II/III, Celeron, Cyrix CPUs, AMD K6's and a lot more to come; and Macintoshs with a range of CPUs; processor speeds ranging from 100 MHz up to 800 MHz; memory capacity of all types. All of the above would yield a large variation in the word error rate (WER) from client to client. With a central server we can control the server configuration and thus make the SR environment and performance predictable and consistent.
3. Although we took into account the client's modest and low data link requirements to be important, we did not realize that other technical benefits were forthcoming from the client-server implementation. By transmitting MFCCs from the client to the server and by doing the SR at the server, one can achieve higher recognition performance (i.e. lower WER) at a fraction of the bit rate than when speech is encoded directly. Contrast this with the client-server model using MFCCs shipped from the client to the server, where the WER is

6.5% at a bit rate of only 3900 bps. The bit rate can be reduced to 2000 bps without increasing the WER by adjusting the product VQ parameters.

4. The final advantage of using MFCC is that it can allow us to implement switched grammars in the server more easily because the resources – memory, CPU and software code can be centrally located and made to work more effectively.

In summary, I am confident that we have made the right decision to go with the client-server and transmit the MFCC from client to server.

The technique of streaming the MFCCs [using HAPI 2.02's technology] should bring some large advantages since it will allow the recognition to begin <u>before</u> the utterance is complete. This is a big advantage over the server-only or the client-only implementation.

I understand the dilemma you are facing: Why do we have to load up or install the same software on both the client and the server and not do the recognition on the client when all the software resources are already there. There is one subtle reason which is mentioned in your discussion – that is - the data file to represent the MFCC's is certainly smaller that the speech data that it represents. This is obvious and we should be able to send the MFCC file to the server faster than if we had larger size file. This should also allow us to improve the latency etc. SRI's paper also hints at the fact that there is more redundancy in the MFCC representation and therefore it is one of the reasons why there is a lower WER for the client-server implementation.

The central reason at the time of our decision goes back to the lack of control of the resources that are available on the client. Now at this point we have access to SRI's data, I think that our decision is validated.

But there is still the key question of how do we segment Entropic's software into client and server. I agree that there is no easy way – since Entropic did not start out assuming a client-server model. Their business and technical model is user- or client-centric based. This is not the way the Internet works – where we may consider the Internet to be big client server system. I suggest that we are aware of the waste of resources by having all of the Entropic SR code duplicated on the client, but we do not know how to remove the unwanted functions. However, we should keep in mind that the streaming the MFCC vectors to the server from the client at a lower data rate [for the largest segment of the user base] should allow better performance than the other approaches.

Finally, we are trying to combine speech recognition with natural language processing. For natural language processing to work well we have to achieve the <u>lowest</u> WER.

Subj: **Re: Preliminary Patent Documents Sent 8/16**
Date: 8/16/99 10:39:48 AM Pacific Daylight Time
From: imb@best.com (I. Bennett)
To: NicdaGreek@aol.com
CC: imb@best.com (Ian Bennett)

Hi Nick:

Glad you received the documents intact and in spite of AOL's problems.

You seem to have everything. Please ignore the 2nd page of the client-side Visio drawing - it should not have been included.

As I mentioned, I will be sending you more docs between now and the time we meet.

Have a great holiday, and looking forward to meeting you in September.

Regards, Ian

On Mon, 16 Aug 1999 NicdaGreek@aol.com wrote:

> Hi Ian,
>
> I got your documents this a.m. I just wanted to confirm their contents:
>
> Spec: 22 pages
> patent draft: 9 pages
> reasons: 2 pages
>
> Client side: 2 pages of drawings (for some reason, page 2 did not come across well; there is a library I seem to be missing my version of VISIO)
>
> Server side: 1 drawing page
>
> Let me know if this is intact. I was lucky to get them, because AOL is mostly down today; what perfect timing. They are rarely down, except the days I have to leave...
>
> I will look them over on my way to/from Europe; talk to you in September.
>
> NICK GROSS
>

---------------------- Headers --------------------------------
Return-Path:
Received: from aol.com (rly-zd03.mail.aol.com [172.31.33.227]) by air-zd01.mail.aol.com (v60.25) with ESMTP; Mon, 16 Aug 1999 13:39:47 -0400
Received: from shell11.ba.best.com (shell11.ba.best.com [206.184.139.142]) by rly-zd03.mx.aol.com (v60.25) with ESMTP; Mon, 16 Aug 1999 13:39:46 -0400
Received: from localhost (imb@localhost)
by shell11.ba.best.com (8.9.3/8.9.2/best.sh) with ESMTP id KAA08819;
Mon, 16 Aug 1999 10:39:45 -0700 (PDT)
Date: Mon, 16 Aug 1999 10:39:44 -0700 (PDT)
From: "I. Bennett"
To: NicdaGreek@aol.com
cc: Ian Bennett
Subject: Re: Preliminary Patent Documents Sent 8/16
In-Reply-To: <50e942aa.24e99e39@aol.com>
Message-ID:
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII

Subj: **Re: patent samples**
Date: 9/6/99 3:04:51 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

Nick:

Thanks for the references you sent. I was able to download the 2 patents from the IBM site and will poke around a bit and elsewhere to do some searching.

That's one area we didn't talk much about this morning - do you think we need a good search for work done in the natural language/speech recognition area? Please let me know.

Anyway, thanks for your time for the meeting and I look forward to working with you and meeting my schedule for the patents.

Regards, Ian

At 03:57 PM 9/6/99 -0400, you wrote:
>Hi Ian,
>
>Our high-speed connection (DSL) is not working, so it would take forever for
>me to send you a reasonable sized PDF; consequently, I am going to send it to
>you from my house.
>
>The samples I was thinking of are:
>
>U.S. Patent No. 5,758,322
>U.S. Patent No. 5,602,963
>
>If you want to get a head start, you can also download them (for a $3 fee)
>yourself directly from the IBM website in PDF format; otherwise, wait till
>tonight.
>
>Here is the URL: http://patent.womplex.ibm.com/cgi-bin/patsearch
>
>This site also has some primitive search logic, and I encourage you to do
>some poking around as well to see what else is out there. This exercise
>helps to serve as foundational material, and helps avoid stepping on any land
>mines that may exist.
>
>Best regards,

>
>NICK GROSS

---------------------- Headers -------------------------------
Return-Path:
Received: from rly-yh02.mx.aol.com (rly-yh02.mail.aol.com [172.18.147.34]) by air-yh04.mail.aol.com (v60.28) with ESMTP; Mon, 06 Sep 1999 18:04:51 -0400
Received: from proxy4.ba.best.com (proxy4.ba.best.com [206.184.139.15]) by rly-yh02.mx.aol.com (v60.28) with ESMTP; Mon, 06 Sep 1999 18:04:46 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy4.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id PAA02362
for ; Mon, 6 Sep 1999 15:02:44 -0700 (PDT)
Message-Id: <4.2.0.58.19990906145033.00a48e80@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Mon, 06 Sep 1999 15:04:48 -0700
To: NicdaGreek@aol.com
From: Ian Bennett
Subject: Re: patent samples
In-Reply-To:
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"; format=flowed

IBM to Unveil Speech-Recognition Software That's Easier to Use

Armonk, New York, Sept. 7 (Bloomberg) -- International Business Machines Corp., the world's largest computer maker, tomorrow will unveil an easier-to-use version of its speech- recognition software, in an effort to broaden its appeal.

The new ViaVoice software commits 30 percent fewer errors in transforming dictation into text than does IBM's current program, the company said. With the application, users can move a computer cursor with verbal commands, eliminating the need for a mouse.

IBM's hope is to make speech recognition as common as a keyboard in communicating commands to a computer. Customers asked IBM to upgrade its technology to make it easier to edit documents and train the software to respond to a user's voice.

The software can accurately record up to 99 percent of the material dictated, IBM said.

The new software links to the Internet so the user can voice commands to scan the World Wide Web, send e-mail or conduct online chats. The program also will read aloud a Web page's content.

Three versions of ViaVoice will be sold, ranging from $59.95 to $179.

Sep/07/1999 22:44

For more stories from Bloomberg News, click here.

Subj: **Progress with Patent Docs**
Date: 9/12/99 8:46:14 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

Hi Nick:

We have been very busy and making good progress with the patent docs. Over the last few days the team in Bangalore finalized the drawings and supplied me with textual descriptions of each block for both the client- and server-side logic.

I am reformatting, labelling, cleaning up the text etc. for it to conform to the outline that you have. I will be definitely be able to send the server-side description by tomorrow.

Could you let me know when will be the last time you will be checking your mail before your departure on Tuesday 9/14?

The docs for the other patent - DB Application - have also been prepared. I need to format this as well.

Regards, Ian

---------------------- Headers -------------------------------
Return-Path:
Received: from rly-zd01.mx.aol.com (rly-zd01.mail.aol.com [172.31.33.225]) by air-zd01.mail.aol.com (v60.28) with ESMTP; Sun, 12 Sep 1999 23:46:14 -0400
Received: from proxy3.ba.best.com (proxy3.ba.best.com [206.184.139.14]) by rly-zd01.mx.aol.com (v60.28) with ESMTP; Sun, 12 Sep 1999 23:46:08 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy3.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id UAA09738
for ; Sun, 12 Sep 1999 20:45:36 -0700 (PDT)
Message-Id: <4.2.0.58.19990912203659.00a5b100@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Sun, 12 Sep 1999 20:47:05 -0700
To: NicdaGreek@aol.com
From: Ian Bennett
Subject: Progress with Patent Docs
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"; format=flowed

Hi Ian,

Got the materials; I will take with me on my trip tomorrow.

Unfortunately my laptop has died a premature death, and the new one is not to arrive for a few days, so I am without access for a few days in Taiwan; I hope to be back online by Saturday.

I will give you feedback when I can.

Best regards,

NICK GROSS

Subj: **Docs for Patent on 9/13/99**
Date: 9/13/99 10:02:05 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

File: DocsforP.mim (730748 bytes)
DL Time (TCP/IP): < 2 minutes

This message is a multi-part MIME message and will be saved with the default filename DocsforP.mim
----
9/13/99

Hi Nick:

I am attaching the following documents:

Word files:
* NLQS TOC (Table of Contents)
* NLQS_1_0 (Field of Invention)
* NLQS_2_0 (Background of Invention)
* NLQS_3_0 (Summary of Invention)
* NLQS_4_0 (List of Drawings and Tables)
* NLQS_5_2 (Client-side Description)
* NLQS_5_3 (Server-side Description)

Visio diagrams:
* Client-side logic (4 pages)
* Server-side logic (4 pages)

As I mentioned in my previous e-mail, I have most of the description written for the software flow. This is the first pass so I will be improving it as I iteratively redo portions. Maybe the way in which the diagrams are labeled are not acceptable. Please let me know any and all criticisms.

The docs for the other patent are all written but I need to organize it and send it in the next couple of mails.

I have decided to save each file separately (with notation like section number and description). This will help us to organize it. Also I have put the revision date in the left footer.

I will be sending you more stuff as quickly as I can get it together.

I think you said that you would have Internet/e-mail access while you are away.

Regards, Ian

----------------------- Headers --------------------------------
Return-Path:
Received: from rly-yg02.mx.aol.com (rly-yg02.mail.aol.com [172.18.147.2]) by air-yg02.mail.aol.com (v60.28) with ESMTP; Tue, 14 Sep 1999 01:02:04 -0400
Received: from proxy4.ba.best.com (proxy4.ba.best.com [206.184.139.15]) by rly-yg02.mx.aol.com (v60.28) with ESMTP; Tue, 14 Sep 1999 01:01:33 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy4.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id VAA26205
for ; Mon, 13 Sep 1999 21:58:19 -0700 (PDT)
Message-Id: <4.2.0.58.19990913215209.00a73580@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Mon, 13 Sep 1999 22:00:05 -0700
To: NicdaGreek@aol.com
From: Ian Bennett
Subject: Docs for Patent on 9/13/99
Mime-Version: 1.0
Content-Type: multipart/mixed;
boundary="====================_223755863==_"

NATURAL LANGUAGE QUERY SYSTEM

TABLE OF CONTENTS

1.0 FIELD OF INVENTION

This invention relates to a system that implements a fully interactive method for answering user's questions. This interactive system is implemented over the worldwide web (WWW) services of the Internet so that when an end user articulates his or her question to the integrated or attached microphone of the user's personal computer or workstation, then after a short period of time an a single answer that corresponds to the best answer is returned to the user in the form of speech in the same natural language that the user presented the question.

This system can be adapted to several applications that require answers to a user's question. Specifically this system is suitable for answering typical student's questions that follow the tutorial presentation or reading of material of a section or module of an on-line course. Questions that relate to the current section or module or chapter or the entire course can be presented to this system. Once the student selects the area for which the question refers to, the system uses this as an environmental variable in its processing and presents the user with the "correct" answer after some short latency time. An agent in the form of a character that mimics the personality of a teacher is used to converse with the user. In addition, this character carries out limited conversation with the user to provide an interactive environment. Another application is in e-commerce web sites. The character can function as a sales agent by answering relevant pre-sales questions such as about stock at hand of particular items, or providing specific single answers relating to the search for a particular item for example a book by its title, its author or other user reference.

[First Draft – to be revised]

2.0 BACKGROUND OF INVENTION

Talking Points and Rough Notes:

- Discuss the nature of the invention and the problem that it solves in applications such as on-line training, customer service systems such as web sites that now list columns of questions and answers, e-commerce sites, remote diagnostic question/answer environments that require answers to user's questions etc. etc.

- Explain how with the rise of the Internet, there is a lack of customer interaction with the typical web site –specifically e-commerce sites. It is usually a point-and-click and then another point-and-click. Usually this is a linear process that requires the user to traverse several lists in order to find the correct information. This invention solves the problem of linearly searching through several lists and links on a particular site.

- Discuss also the role of the agent and how the character is used to enhance interactivity – with speech recognition at the server. Discuss in particular the evolution of speech recognition technology so far, and how distributed SR results in – controlled and predictable process that is also scaleable.

- Talk about how speech recognition systems have evolved up to this point – single platform – non-distributed, i.e. non-client/server implementations. Now with the increase in processing power, distributed processing or client/server based SR systems are possible and meaningful as this development shows. Discuss how important it is to have maximum SR or minimum WER when combining SR with NL technologies. How do we achieve this – by using HMMs, small grammars, dynamically switched grammars and dictionaries etc. Also we trade of the extraction of the minimum number of extracted MFCC vectors at the client with the transit time required to send to server. We allow the server to extract the other derived vectors – delta and acceleration parameters. Breakthroughs in processor power and linguistic analysis combined with novel and simple yet robust algorithms make the NLQS implementation possible.

- This is possibly the section where we discuss in detail the mechanics of SR and the linguistic processing that we have developed for this invention.]

3.0 SUMMARY OF INVENTION

This invention relates to a natural language query system (NLQS) and the implementation of a fully interactive method for answering user's questions over distributed network such as the Internet or a local intranet. This interactive system when implemented over the worldwide web (WWW) services of the Internet functions so that a client or user can ask a question in a natural language such as English, French, German or Spanish and receive the appropriate answer at his or her personal computer also in his or her native natural language. The question that is asked at the client's machine is articulated by the speaker and captured by a microphone that is built in as in the case of a notebook computer or is supplied as a standard attachment. Once the question is captured, the question is processed partially by NLQS client-side software resident in the client's machine. The output of this partial processing is a set of speech vectors which are transported to the server via the Internet to complete the recognition of the user's questions. This recognized speech is then converted to text at the server.

After the user's question is decoded by the speech recognition engine located at the server, the question is converted to a structured query language (SQL) query. This query is then presented to a software process within the server called the DBProcess.dll for preliminary processing and to the Natural Language Engine (NLE) module for extracting the noun phrases (NP) of the user's question. During the process of extracting the noun phrase within the NLE, the tokens of the users' question are tagged. The tagged tokens are then grouped so that the NP list can be determined. This information is stored and sent to the DBProcess.

In the DBProcess.dll, the SQL query is fully customized using the NP extracted from the user's question and other environment variables that are relevant to the application. For example, in a training application, the user's selection of course, chapter and or section would constitute the environment variables. The SQL query is constructed using the SQL several Full-Text predicates such as CONTAINS, NEAR, AND operators. The SQL query is next sent to the Full-Text search engine within the SQL database, where a Full-Text search procedure is initiated. The result of this search procedure is recordset of answers. This recordset contains stored questions that are similar linguistically to the user's question. Each of these stored questions has a paired answer stored in a separate text file, whose path is stored in a table of the database.

The entire recordset of returned stored answers is then returned to the NLE engine in the form of an array. Each stored question of the array is then linguistically processed sequentially one by one. This linguistic processing constitutes the second step of a 2-step algorithm to determine the single best answer to the user's question. This second step proceeds as follows: for each stored question that is returned in the recordset, a NP of the stored question is compared with the NP of the user's question. After all stored questions of the array are compared with the user's question, the stored question that yields the maximum match with the user's question is selected as the best possible stored question that matches the user's question.

The stored answer that is paired to the best-stored question is selected as the one that answers the user's question. The ID tag of the question is passed to the DBProcess.dll. This DBProcess.dll the returns the answer which is stored in a file.

[to be revised – not complete]

4.0 LIST OF DRAWINGS AND TABLES

The following are the list of drawings:

The following are the list of tables:

5.3 SERVER-SIDE DESCRIPTION

5.3.1 The Server System

The server-side logic of the Natural Language Query System consists of the following two dynamic link libraries (DLLs):

1. CommunicationServerISAPI.dll labeled “1” in Server-side –System Logic diagram
2. DBProcess.dll labeled “2” in Server-side –System Logic diagram

The CommunicationServerIASPI.dll is comprised of 3 modules:

1. The Server-side Speech Recognition Engine
2. Interface module between the Natural Language Engine and the DBProcess.dll
3. Natural Language Engine

The DB Process.dll with label “2”, is a module whose primary function is are to connect to a SQL database and to execute the SQL query. In addition, it interfaces with logic that fetches the correct answer from a file path once this answer is passed to it form the Natural Language Engine.

5.3.2 Speech Recognition Server

The speech recognition engine at the server-side is the distributed component of the speech recognition engine. This is shown in Fig. - Server-side Distributed Component of Speech Recognition Engine. Referring to this diagram, the flow of the signal is as follows:

Within block “1”, the received binary MFCC vector byte stream is received from the client. The MFCC are decoded.

Since the MFCC vectors contains embedded NULL characters, they cannot be transferred to the server as such using http protocol. Thus the MFCC vectors are encoded at the client-side before transmission in such a way that all the data converted into a stream of bytes without embedded NULL characters in the data. At the veruy end of the byte stream a single NULL character is introduced.

Block # 1b

Prepare Dictionary & Grammar file name

The block marked with label ”5” is where the Course, Chapter and/or section selected for asking questions is received. For the speech recognition to be implemented, there has to be a both grammar and dictionary files.

The Grammar file supplies all the sentences that are to be recognized will be provided. While the Dictionary file provides phonemes (the information of how to pronounce a word) of each word in the grammar file.

If we have all the sentences to be recognized for the total system, then the recognition accuracy will be significantly deteriorated and the speed of loading the grammar and dictionary files would slow down the speech recognition process.

To avoid these problems the specific grammars are loaded or actively to be the current grammar according the Course, Chapter and/or Section. Thus the grammar and dictionary

files are loaded dynamically according to the given Course, Chapter and/or Section as dictated by the user.

The second block labeled "2" implements the initialization of the Speech Recognition engine. The MFCC vectors along with the grammar filename and the dictionary file names are introduced to this block to initialize of the speech decoder.

The initialization process consists of the following processes:

- Loading the SRE library – "2a". This then allows the creation of an External Source "2b"using MFCC vectors received. It allocates the memory required to hold the recognition objects "2c", and also creates and initializes the objects "2d" that are required for the recognition such as Source, Coder, Recognizer and Results
- Loading of the Dictionary "2e" and Hidden Markov Models (HMMs) "2f"
- Loading of the Grammar file as shown in "2g".

Speech Recognition is the next step as shown in Block labeled "3". Using the functions created in the External Source, this block reads MFCC vectors one at a time from the External Source "3a", and processes them in block "3b" to realize the words in the speech pattern that are symbolized by the MFCC vectors captured at the client.

Once the speech is recognized, the SRE is un-initialized as shown in block "4" - **Uninitialize SRE.** In this block all the objects created in the initialization block are deleted – label "4a"and memory allocated in the initialization block during the initialization phase are removed – label "4b".

5.3.3 Database Processor – DBProcess DLL

Construction of the SQL Query – Block "10" - To customize the SQL Query for fetching the best suitable question stored in the database, the number of words present in each Noun Phrase of Question asked by the user is first calculated, and segregated and then and stored into an array. Using all the words present in the Noun Phrase along with Database name (**Course**), Table name (**Chapter** and/or **Section**) chosen by the user at client side, the SQL Query is constructed. It also uses full-text predicate **CONTAINS** and the full-text constructs **NEAR( )** and the **AND** operator while preparing the SQL Query**.** This customized SQL Query is then passed to the DBProcess.DLL so that the paired questions can be retrieved from the database.

Connection to the SQL Server - Block "11" implements the Connection to the Database. The connection sequence and the subsequent retrieved recordset is implemented in the following steps:

1. A server name is assigned to the DBProcess.dll member variable
2. A database name is assigned to the DBProcess.dll member variable
3. Connect the SQL Server database
4. Execute the SQL query
5. Extract the total number of records retrieved by the query
6. Allocate the memory to store the total number of paired questions
7. Store the entire number of paired questions into an array

In connecting to the SQL Server itself and to the database that is specified in the code of the DBProcess.DLL, the following 3 steps are executed:

1. The connection object is prepared
2. The connection string is prepared using the database name, user name, password, and the server name.
3. Then a connection is made to the SQL Server database.

The next step is the Execution of the SQL Query. To execute the SQL Query, the SQL Query constructed in Block "10" is passed to Block "12a". After assembling there, it is passed to block "12b" where is executed.

Then there is the creation of code in this same block "12b" that creates a recordset object so that it accommodate the recordset after it is returned from the NLQS database.

[At this point we have to describe the SQL Full-Text Search in detail]

Then the recordset containing paired questions is passed again to the NLE. This step is the 2nd step of the 2-step algorithm. Its carries out detailed linguistic analysis of the paired questions and further it iteratively compares the NP of each record of the recordset with that of the NP of the user's question to identify the stored question that best matches the user's question. The criterion is the NP – the paired question that has the maximal number of NP is chosen to be the one that best matches the user's question.

After the recordset is returned from the database, and the the array of paired questions are stored as an array, this array is then passed to Block "14" of the NLE. The sequence of events that follow are similar to that used to extract the NP of the user's question. In this analysis, the noun phrases are extracted from each of the paired questions in sequence.

The entry point for the array of paired questions to the NLE is port "14" of the NLE. The processing steps that follow are via several sub-blocks labeled "9a" through "9e". Each of these sub-blocks implement the several different linguistic functions performed by the NLE that are required to extract the noun phrases from the array of paired questions. The steps are as follows:

<u>Initialize Grouper Resources Object and the Library</u> – Block "9a" – this block initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

<u>Tokenizing of the words from the given text</u> – block "9b" – in this block all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

<u>The Tagging of all the tokens</u> are implemented in Block "9c". Here the Tagger object will tag all the tokenized words and the output is passed to Grouper object.

<u>The Grouping of all tagged token to form NP list</u> – block "9d, in this block the Grouper object groups all the tagged token words and gives out as Noun Phrases.

Uninitializing of the grouper resources object and freeing of the resources -- block "9e" – in this block all the initialized resources are un-initialized and the resources are freed. Also de-allocates the memory allocated to store all Noun Phrases

At the end of this processing, the NP list is returned from the NLE to block "13" of the diagram "Interface between NLE and DBProcess.dll". Specifically, it is passed to the block "15" – Get Best Answer ID". Before it extracts the ID for the single best answer, processing passes to Block "15a" where a comparison of the NP of user's question with NP of the Paired Questions extracted from database is made to find out the best suitable question present in the database that matches the user's question.

1. This block "15" then passes the "best Answer ID" to block "16" of the Diagram "DBProcess.dll".

Some preparatory steps are executed before processing takes place in block "16". These steps are as follows:

a. Creates a database process DLL class object
b. Gets the NP list from the question and builds the full-text SQL query using the NP list from the question and section name
c. Gets all the paired questions related to the user's questions from the database. If no records are fetched, then an error is returned. If only one record is returned, then the answer is returned using the answer's file path from the fetched single record. Otherwise, the NLE is called to get the single best record.

2. Following this diagram, block "16a" of this diagram receives the "best record number". The file path for this "best record number" is then fetched in block "16b". Next that particular file is opened in block "16c". Also in the same block "16c", the contents of the file are read and the contents passed to block "16d".

In determining the file path, the following are the steps:

1. The record pointer is moved to the specified record number.
2. The path of the answer file is next retrieved
3. The content from the file is read
4. The answer is returned

Once the answer is returned, the file is sent to block "16d", where it is compressed. The actual sequence of steps involved are as follows:

1. The size of the answer to be compressed is obtained
2. Sufficient memory is allocated to hold the compressed answer.
3. The compressed answer and the size of the original answer is returned to the client.

5.3.4 Interface between NLE & DBProcess DLL

This part of the code contains functions, which interface processes resident in the NLE block and DBProcess.dll blocks. The functions are illustrated diagrammatically in **Interface Logic between Natural Language Engine (NLE) and DBProcess DLL.**

Block "8" implements functions that extracts the Noun Phrase (NP) list from the user's question. This part of the code is a function, which interacts with the NLE and gets the list

of Noun Phrases in a sentence. This function is used to find out the Noun Phrases existed in the user query.

Block "13" retrieves the NP list from the list of paired questions and stores these question sinto an array. This part of the code is a function, which interacts with the NLE and gets the list of Noun phrases in a sentence. This function is used to find out the Noun phrases existed in the paired questions that are similar to the user question.

Next the function - **Get Best Answer ID** is implemented as shown in block "15". This part of the code gets the best answerID to the user's query. This function first finds out the number of Noun phrases that are matching with the Noun phrases in the user's query. Then it selects the record, which contains the maximum number of matching Noun phrases.

Then a Comparison of the number of NP in the User's question with that of NP list of paired questions is implemented in block "15a" . This part of the code is a function which is used by block "15a" of the code to compare the find out the number of matching Noun Phrases in the given user's query and a given paired question which is similar to the user asked question.

The ID corresponding to the best answer is then returned to the **Get Best Answer ID** block – "15", which then returns it to the **DBProcess.dll** block.

5.3.5 The Natural Language Engine

This engine implements a number of functions associated with linguistic processing. The key purpose of this block is to refine the search for the best answer so that a single best answer is provided for the user's question.

The entry point of NLE is Block "9" which in turn is made up of several sub-blocks labeled "9a" through "9e". Each of these sub-blocks implement the several different functions required in the NLE.

<u>Initialize Grouper Resources Object and the Library</u> – Block "9a" – this block initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

<u>Tokenizing of the words from the given text</u> – block "9b" – in this block all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

<u>The Tagging of all the tokens</u> are implemented in Block "9c". Here the Tagger object will tag all the tokenized words and the output is passed to Grouper object.

<u>The Grouping of all tagged token to form NP list</u> – block "9d, in this block the Grouper object groups all the tagged token words and gives out as Noun Phrases.

<u>Uninitializing of the grouper resources object and freeing of the resources</u> -- block "9e" – in this block all the initialized resources are un-initialized and the resources are freed. Also de-allocates the memory allocated to store all Noun Phrases

5.2 CLIENT-SIDE DESCRIPTION

5.3.1 The Client System

The client-side logic of the Natural Language Query System as shown in "Client-side – System Logic Diagram" consists of three main processes as follows:

1. Initialization process labeled as "A"
2. Iterative process consisting of two sub-processes: a) Receive User Speech and b) Receive Answer from Server – labeled as "B"
3. Un-initialization process – labeled as "C"

Each of these three processes are described in detail in the following paragraphs.

Initialization at the Client-side

The initialization of the client side as shown in the diagram "Initialization at the Client Side" is comprised of initializing 3 processes: the client-side Speech Recognition Engine – label "A", the MS Agent – label "B" and the Communication processes – label "C".

Initialization of the Speech Recognition Engine

In block "1" the SRE Library is initialized, and memory is allocated to hold the objects of Source and Coder "1a". Source and Coder objects are created – "1b". Loading of configuration file - "1c" also takes place, when the SRE Library is initialized – "1c". In the configuration file, the type of the input of Coder, the type of the output of the Coder are declared.

Next, the Speech and Silence components of an utterance are calibrated – "2". To calibrate the speech and silence components, the user articulates a sentence that is displayed in a text box on the screen. Then the SRE library estimates the noise and other parameters required to find the silence and the speech.

Initialization of the MS Agent

Initialization is described by block "B" of the "Client-side Logic" diagram. This initialization consists of the following steps:

1. Initialize COM library – block "1". This part of the code initializes the COM library, which is required to use the ActiveX Controls.
2. Create instance of Agent Server - block "2" - this part of the code creates an instance of Agent ActiveX control.
3. Loading of the MS Agent – block "3" - this part of the code loads MS Agent character from the specified file.
4. Get Character Interface – block "4" - this part of the code gets the interface for the specified character.
5. Add Commands to Agent Character Option – block "5" - this part of the code adds commands to Agent Properties sheet, which can be accessed by clicking on the icon that appears in the system tray, when the character is loaded e.g: Speak, TTS Properties.
6. Show the Agent Character – block "6" - this part of the code displays the character on the screen
7. AgentNotifySink – block "7" - to handle events. This part of the code creates AgentNotifySink object, registers it and get agent properties interface.

8. Do Character Animations – block "8" - This part of the code plays specified character animations to welcome the user to the NLQS.

The above then constitutes the entire sequence required to initialize the MS Agent.

Initialization of the Communication Process

The initialization of the Communication Process shown as label "C" in the of the "Client-side Logic" diagram. This initialization consists of the following components in the diagram "Initialization at Client-Side":

1. Open Internet Connection – block "1" -this part of the code opens Internet Connection and sets the parameter for the connection.
2. Set Callback Status - block "1" -this part of the code sets the callback status – to inform the user of the status of connection.
3. Start new HTTP Internet session – block "2" this part of the code starts a new Internet session.

Iterative Processing

Once the initialization is complete, an iterative process – see diagram "Iterative Process", is launched when the user presses the Start Button to initiate a query. The iterative process consists of two sub-processes: **Receive User Speech** and the **Receive User Answer.** The **Receive User Speech** receives speech from the user, while the **Receive User Answer** receives the answer to the user's question in the form of text from the server so that it can be converted to speech for the user.

Receive User Speech – The SRE and the Communication processes are the 2 processes involved in receiving the user's utterance. The coder is prepared – block "1" so that the Coder object receives speech data from the source object. Next the Source Start is initiated – block "2". This part of the code initiates the data retrieval using the source Object which will in turn be given to the Coder object. Next MFCC vectors are extracted from the Speech utterance – block "3" until silence is detected.

Communication – the communication module is used to implement the transport of data from the client to the server over the Internet. The data consists of encoded MFCC vectors that will be used at then server-side of the Speech Recognition engine to complete the wpeech recognition decoding. The sequence of the communication is as follows:

1. **OpenHttprequest** - block "1" - this part of the code first converts MFCC vectors to a stream of bytes and processes the bytes so that it is compatible with the HTTP protocol.

2. **Encode the MFCC Byte Stream** – block "2" - this part of the code encodes MFCC vectors, so that it can be sent to the server via http. (Regarding the encoding more information is given in the server side document)
3. **Send the data** – block "3" - this part of the code sends MFCC vectors to the server using the Internet connection and the http protocol.
4. **Wait for the Server Response – block "4" -** this part of the code sends will be in the loop till the response from the server arrives

Receive Answer from Server – the MS Agent – labels "1" and "2", Text-to-Speech Engine – labels "3" and "4" and the Communication modules – labels "5", "6" and "7"are the 3 processes that are involved in the Receive Answer from Server as shown in the diagram "Iterative Process". The MS Agent.

MS Agent 1.Speak

Block #1
Receive the answer

Description: This part of the code receives the answer form communication part

Block #2
Speak the answer received from the server

Description: The MS Agent will articulate it using the TTS.

Communication: InternetRead

Block #1

Receive the best answer from the server in a compressed form

Description: This part of the code encodes receives the answer from the server

Block #2

Uncompress the received answer

Description: This part of the code uncompresses the received answer. Then it will be passed on to the MS Agent module

Block #3

Pass the answer to the MS agent

Description: The MS Agent will articulate the answer using the TTS.

Uninitialization

SRE

Block #1
Delete the created objects

Description: This part of the code deletes the created objects during initialization and deallocates the memory allocated in the initialization process

Communication

Block #1

Close the Internet Session

Description: This part of the code closes the Internet session and Internet connection.

Block #2

Close the Internet Connection

Description: This part of the code closes the Internet connection

MS Agent

Block #1

Release Commands Interface

Description: This part of the code releases the commands added to property sheet during loading of character

Block #2

Release Character Interface

Description: This part of the code releases the character

Block #3

Unload Agent

Description: This part of the code unloads the character

Block #4

Release the AgentNotifysink Interface

Description: This part of the code releases the sink object

Block #5

Release property sheet Interface

Description: This part of the code releases the property sheet

Block #6

Unregister Agent

Description: This part of the code unregisters the Agent

Block #7

Release the Agent Interface

Description: This part of the code releases all the resources allocated during initialization

Subj: **Draft Patent Docs for the NLQS Database Application**
Date: 9/19/99 12:49:19 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

File: DraftPat.mim (579074 bytes)
DL Time (TCP/IP): < 1 minute

This message is a multi-part MIME message and will be saved with the default filename DraftPat.mim
----
Hi Nick:

Hope you had a successful trip. I hope your jet lag is not as bad as I experienced when I returned from my last Asian trip.

I did not send any additional stuff while you were away.

However, I am sending you the first draft of the 2nd patent. There is still much writing to do for this one even though it is conceptually less complex. I decided to send this so that you have an better idea of the scope of the application.

I am expecting to receive the actual application from Bangalore by tonight and I will then be able to have it open on my desktop as I write the detailed description of all the functions of the user interface.

Please expect one more mail to follow this which includes another Visio drawing.

Regards, Ian

----------------------- Headers --------------------------------
Return-Path:
Received: from rly-yb02.mx.aol.com (rly-yb02.mail.aol.com [172.18.146.2]) by air-yb05.mail.aol.com (v60.28) with ESMTP; Sun, 19 Sep 1999 15:49:18 -0400
Received: from proxy2.ba.best.com (proxy2.ba.best.com [206.184.139.14]) by rly-yb02.mx.aol.com (v61.9) with ESMTP; Sun, 19 Sep 1999 15:48:59 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy2.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id MAA20946
for ; Sun, 19 Sep 1999 12:45:52 -0700 (PDT)
Message-Id: <4.2.0.58.19990919123606.00a8e570@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Sun, 19 Sep 1999 12:48:01 -0700
To: NicdaGreek@aol.com

From: Ian Bennett
Subject: Draft Patent Docs for the NLQS Database Application
Mime-Version: 1.0
Content-Type: multipart/mixed;
boundary="=====================_227837978==_"

Subj: **Draft Patent Docs for the NLQS Database Application -**
Date: 9/19/99 1:04:19 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

File: User Interface for NLQS Database Application (12 pages).vsd (643584 bytes)
DL Time (TCP/IP): < 1 minute

Hi Nick:

Attached is the first draft of the User Interface diagram for the NLQS Database Application. It's a Visio diagram with 12 pages.

I intend add labels to each screen shot and refer to each label as I document the detailed description.

Regards, Ian

----------------------- Headers --------------------------------
Return-Path:
Received: from rly-zb05.mx.aol.com (rly-zb05.mail.aol.com [172.31.41.5]) by air-zb02.mail.aol.com (v60.28) with ESMTP; Sun, 19 Sep 1999 16:04:18 -0400
Received: from proxy2.ba.best.com (proxy2.ba.best.com [206.184.139.14]) by rly-zb05.mx.aol.com (v61.9) with ESMTP; Sun, 19 Sep 1999 16:01:02 2000
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy2.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id MAA05515
for ; Sun, 19 Sep 1999 12:54:50 -0700 (PDT)
Message-Id: <4.2.0.58.19990919125218.00a908c0@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Sun, 19 Sep 1999 12:56:38 -0700
To: NicdaGreek@aol.com
From: Ian Bennett
Subject: Draft Patent Docs for the NLQS Database Application - Visio
Drawing
Mime-Version: 1.0
Content-Type: multipart/mixed;
boundary="====================_228355073==_"

Subj: **Draft Patent Docs for the NLQS Database Application**
Date: 9/19/99 12:49:19 PM Pacific Daylight Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

File: DraftPat.mim (579074 bytes)
DL Time (28800 bps): < 6 minutes

This message is a multi-part MIME message and will be saved with the default filename DraftPat.mim
----
Hi Nick:

Hope you had a successful trip. I hope your jet lag is not as bad as I experienced when I returned from my last Asian trip.

I did not send any additional stuff while you were away.

However, I am sending you the first draft of the 2nd patent. There is still much writing to do for this one even though it is conceptually less complex. I decided to send this so that you have an better idea of the scope of the application.

I am expecting to receive the actual application from Bangalore by tonight and I will then be able to have it open on my desktop as I write the detailed description of all the functions of the user interface.

Please expect one more mail to follow this which includes another Visio drawing.

Regards, Ian

----------------------- Headers --------------------------------
Return-Path:
Received: from rly-yb02.mx.aol.com (rly-yb02.mail.aol.com [172.18.146.2]) by air-yb05.mail.aol.com (v60.28) with ESMTP; Sun, 19 Sep 1999 15:49:18 -0400
Received: from proxy2.ba.best.com (proxy2.ba.best.com [206.184.139.14]) by rly-yb02.mx.aol.com (v61.9) with ESMTP; Sun, 19 Sep 1999 15:48:59 -0400
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy2.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id MAA20946
for ; Sun, 19 Sep 1999 12:45:52 -0700 (PDT)
Message-Id: <4.2.0.58.19990919123606.00a8e570@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Sun, 19 Sep 1999 12:48:01 -0700
To: NicdaGreek@aol.com

From: Ian Bennett
Subject: Draft Patent Docs for the NLQS Database Application
Mime-Version: 1.0
Content-Type: multipart/mixed;
boundary="=====================_227837978==_"

5.2 CLIENT-SIDE DESCRIPTION

The Client System

The client-side logic of the Natural Language Query System is illustrated in **Fig. 2**. Referring to **Fig. 2**, three main processes are illustrated as follows: Initialization process **200A** consisting of processes SRE **201,** Communication **202** and MS Agent **203;** an iterative process **200B** consisting of two sub-processes: a) Receive User Speech **208** – made up of SRE **204 and Communication 205;** and b) Receive Answer from Server **207** and an un-initialization process **200B**. The un-initialization process is made up of three sub-processes: SRE **212,** Communication **213**, and MS Agent **214.**

Each of the above three processes are described in detail in the following paragraphs.

Initialization at the Client-side

The initialization of the client side as illustrated in **Fig. 2-2** is comprised of initializing 3 processes: the client-side Speech Recognition Engine **220A**, the MS Agent **220B** and the Communication processes **220C**.

Initialization of the Speech Recognition Engine

In block **1** the SRE COM Library is initialized, and memory **220** is allocated to hold the objects of Source and Coder. Source and Coder objects **221**are created. Loading of configuration file **221A** from the configuration file **221B** also takes place at the same time that the SRE Library is initialized. In the configuration file **221B**, the type of the input of Coder, the type of the output of the Coder are declared.

Next, the Speech and Silence components of an utterance are calibrated **222**. To calibrate the speech and silence components, the user articulates a sentence that is displayed in a text box on the screen. Then the SRE library estimates the noise and other parameters required to find the silence and the speech.

Initialization of the MS Agent

Initialization is described by block **220B** as illustrated in **Fig. 2-2**. This initialization consists of the following steps:

1. Initialize COM library **223**. This part of the code initializes the COM library, which is required to use the ActiveX Controls.
2. Create instance of Agent Server **224** - this part of the code creates an instance of Agent ActiveX control.
3. Loading of the MS Agent **225** - this part of the code loads MS Agent character from the specified file **225A**.
4. Get Character Interface **226** - this part of the code gets the interface for the specified character.
5. Add Commands to Agent Character Option **227** - this part of the code adds commands to Agent Properties sheet, which can be accessed by clicking on the icon

that appears in the system tray, when the character is loaded e.g: Speak, TTS Properties.

6. Show the Agent Character **228** - this part of the code displays the character on the screen
7. AgentNotifySink - to handle events. This part of the code creates AgentNotifySink object **229**, registers it **230** and Get Agent Properties interface **231**. The property sheet to the Agent character is done in **232**.
8. Do Character Animations **233** - This part of the code plays specified character animations to welcome the user to the NLQS.

The above then constitutes the entire sequence required to initialize the MS Agent.

Initialization of the Communication Process

The initialization of the Communication Process **220C** is also illustrated in **Fig. 2-2**. Referring to **Fig. 2-2**, this initialization consists of the following components: Open Internet Connection **234** - this part of the code opens Internet Connection and sets the parameter for the connection. Then the Set Callback Status **235** sets the callback status so as to inform the user of the status of connection. Finally Start New HTTP Internet Session **236** starts a new Internet session.

Iterative Processing

As illustrated in **Fig. 3**, once the initialization is complete, an iterative process is launched when the user presses the Start Button to initiate a query. Referring to **Fig. 3**, the iterative process consists of two sub-processes: **Receive User Speech 240** and the **Receive User Answer 243.** The **Receive User Speech 240** receives speech from the user, while the **Receive User Answer 243** receives the answer to the user's question in the form of text from the server so that it can be converted to speech for the user by the text-to-speech engine.

Receive User Speech – As illustrated in **Fig. 3**, the SRE **241** and the Communication **242** processes are the two (2) processes involved in receiving the user's utterance. The coder **248** is prepared so that the coder object receives speech data from the source object. Next the Start Source **249** is initiated. This part of the code initiates the data retrieval using the source Object which will in turn be given to the Coder object. Next MFCC vectors **250** are extracted from the Speech utterance until silence is detected.

Communication – the communication module **242** is used to implement the transport of data from the client to the server over the Internet. The data consists of encoded MFCC vectors that will be used at then server-side of the Speech Recognition engine to complete the speech recognition decoding. The sequence of the communication is as follows:

1. **OpenHTTPRequest 251** - this part of the code first converts MFCC vectors to a stream of bytes and processes the bytes so that it is compatible with the HTTP protocol.

2. **Encode the MFCC Byte Stream 251** - this part of the code encodes MFCC vectors, so that it can be sent to the server via http. (Regarding the encoding more information is given in the server side document)
3. **Send the data 252** - this part of the code sends MFCC vectors to the server using the Internet connection and the http protocol.

4. **Wait for the Server Response 253 -** this part of the code sends will be in the loop till the response from the server arrives

Receive Answer from Server 243 is comprised of the following modules as shown in **Fig. 3**.: the MS Agent **244**, Text-to-Speech Engine **245** and the Communication modules **246**. All three modules interact to receive the answer from the server. As illustrated in **Fig. 3**, the communication process consists of three processes: the Receive the Best Answer **258** receives the best answer over the HTTP communication channel. The answer is de-compressed in **259** and then the answer is passed to the MS Agent by **260**. MS Agent **254** receives the answer from Communication process **246**. MS Agent **255** then articulates the answer using the text-to-speech engine **257**. The text to speech engine uses the natural language voice data file **256** that is associated with it.

Uninitialization
The un-initialization is illustrated in **Fig. 4**. Three functions are un-initialized – the SRE **270**, Communication, **271** and the MS Agent **272**. To un-initialize the SRE, memory that was allocated in the initialization phase is de-allocated **273** and objects created during initialization phase are deleted **274.** As illustrated in **Fig. 4**, to un-initialize the Communication **271**, and the Internet connection previously established with the server is closed **275**. Next the Internet session created at the time of initialization id closed **276.** For the un-initialization of the MS Agent **272,**as illustrated in **Fig. 4**, the Commands Interface is first released **277**. This releases the commands added to property sheet during loading of character. Next the Character Interface is released **278** and the Agent is unloaded **279**. The Sink Object Interface is then released **280** followed by the release of the Property Sheet Interface **281**. The Agent Notify Sink **282** is then un-registered the Agent and finally the Agent Interface **283** is released which releases all the resources allocated during initialization.

5.3 SERVER-SIDE DESCRIPTION

Server System
The server-side of the NLQS system as illustrated in **Fig. 5** is comprised of several key processing modules. These include the Communication module – CommunicationServerISAPI **500**, the database processor DBProcess **501**, the Natural language engine (NLE) **9, 14**, and the interface between the NLE and the DBProcess **500B**. The Communication module – CommunicationServerISAPI **500** includes the server-side speech recognition engine and the communication interfaces required between the client and server.

As illustrated in **Fig. 5**, the server-side logic of the Natural Language Query System as consists of the following two dynamic link library components: CommunicationServerISAPI **500** and the DBProcess **501**. The CommunicationServerIASPI is comprised of 3 modules: the Server-side Speech Recognition Engine **501A**; the Interface module between the Natural Language Engine and the DBProcess. **500B**; and the Natural Language Engine **500C.**

The DB Process **501**, is a module whose primary function is to connect to a SQL database and to execute the SQL query. In addition, it interfaces with logic that fetches the correct answer from a file path once this answer is passed to it from the Natural Language Engine **500C**.

Speech Recognition: Server-Side
The speech recognition engine **500A** is a set of distributed components of the speech recognition engine that reside at the server-side. Referring to **Fig. 4-5-1**, the signal flow for speech recognition at the server-side is as follows:

Within **601**, the received binary MFCC vector byte stream corresponding to the speech signal's acoustic features extracted at the client and sent over the communication channel is received. The MFCC acoustic vectors are decoded from the encoded HTTP byte stream as follows: Since the MFCC vectors contains embedded NULL characters, they cannot be transferred to the server as such using http protocol. Thus the MFCC vectors are encoded at the client-side before transmission in such a way that all the data converted into a stream of bytes without embedded NULL characters in the data. At the very end of the byte stream a single NULL character is introduced.

Dictionary Preparation & Grammar Files
Referring to **Fig. 4-5-1**, within block **605**, the options selected by the user: Course, Chapter and/or section are received. These selected options define the scope – i.e. grammars and dictionaries hat will be dynamically loaded to speech recognition engine for Viterbi decoding. For the speech recognition to be implemented, there has to be a both grammar and dictionary files. The Grammar file supplies all the sentences that are to be recognized will be provided, while the Dictionary file provides phonemes (the information of how to pronounce a word) of each word in the grammar file. If all the sentences for a given environment that can be recognized are contained in a single grammar file then the recognition accuracy will be deteriorated and the speed of loading the grammar and dictionary files would impair the speech recognition process. To avoid these problems,

specific grammars are loaded or actively made to be the current grammar according the Course, Chapter and/or Section selected. Thus the grammar and dictionary files are loaded dynamically according to the given Course, Chapter and/or Section as dictated by the user. The second block **2** implements the initialization of the Speech Recognition engine. The MFCC vectors along with the grammar filename and the dictionary file names are introduced to this block to initialize of the speech decoder.

As illustrated in **Fig. 4-5-1**, the initialization process consists of the following processes: Loading the SRE library **602a**. This then allows the creation of an External Source **602b** using MFCC vectors received. It allocates the memory required to hold the recognition objects **602c**, and also creates and initializes the objects **602d** that are required for the recognition such as Source, Coder, Recognizer and Results Loading of the Dictionary **602e** and Hidden Markov Models (HMMs) **602f**; Loading of the Grammar file as illustrated in **602g**.

Speech Recognition **603** is the next step as illustrated in **Fig. 4-5-1**. Using the functions created in the External Source, this block reads MFCC vectors one at a time from the External Source **603a**, and processes them in block **603b** to realize the words in the speech pattern that are symbolized by the MFCC vectors captured at the client.

Once the user's speech is recognized, the flow of the SRE passes to Un-initialize SRE **604** where the speech engine is un-initialized as illustrated. In this block all the objects created in the initialization block are deleted **604a**, and memory allocated in the initialization block during the initialization phase are removed **604b**.

Database Processor – DBProcess
Construction of the SQL Query as illustrated in **Fig. 4-5-4,** Construct SQL Query using the **SELECT** and **CONTAINS** predicate **950**, is the module that constructs the SQL query used for retrieving the best suitable question stored in the database. The table name is then concatenated with the constructed **SELECT** statement **951**. Next the number of words present in each Noun Phrase of Question asked by the user is first calculated **952**. Then as much memory is allocated **953** for all the words present in the NP. Next the word List present in the NP **954** is obtained. Next, these words are concatenated to the SQL Query separated with the **NEAR ( )** keyword **955**. Next, the **AND** keyword is concatenated to the SQL Query after each NP **956.** Finally the memory resource is freed so as to allocate memory to store the words received from NP for the next iteration **957**.

As illustrated in **Fig. 5**, using all the words present in the Noun Phrase along with Database name (Course), Table name (Chapter and/or Section) chosen by the user at client side, the SQL Query is constructed **10**. It also uses full-text predicate **CONTAINS** and other predicates such as **NEAR( )** and the **AND** operator while preparing the SQL Query. This customized SQL Query is then passed to the DBProcess **12** so that the paired questions can be retrieved from the NLQS database.

Connection to the SQL Server – As illustrated in **Fig. 4-5-2**, **711** implements the Connection to the Database. The connection sequence and the subsequent retrieved recordset is implemented in the following steps:

1. Server and database names are assigned to the DBProcess member variable **711A**
2. Connect the SQL Server database, **711C**
3. The SQL Query is received, **712A**
4. Execute the SQL query, **712B**
5. Extract the total number of records retrieved by the query
6. Allocate the memory to store the total number of paired questions
7. Store the entire number of paired questions into an array

In connecting to the SQL Server itself and to the database that is specified in the code of the DBProcess, the following 3 steps are executed:

1. The connection object is prepared
2. The connection string is prepared using the database name, user name, password, and the server name, **711A**
3. Then a connection is made to the SQL Server database, **711C**

Now referring back to **Fig. 5**, the next step is the Execution of the SQL Query **12B**. To execute the SQL Query, the SQL Query constructed in Block **10** is passed to **712a** of **Fig. 4-5-2**. After assembling there, it is passed to block **712b** where is executed.

Then there is the creation of code in this same block **712b** that creates a recordset object so that it accommodate the recordset after it is returned from the NLQS database.

NLQS Database Table Organization

Figure 7 illustrates structure of tables used in a typical NLQS database. The NLQS database that is used part of the query system required in training environment consists typically of a Course 7**01**, which is made of several chapters **702**, **703**, **704**. Each of these chapters can have one or more Sections **705**, **706**, **707** as shown for Chapter 1. A similar structure can exist for Chapter 2, Chapter 3 … Chapter N.

Each section has a set of one or more question answer pairs **708**, **709**, **710** stored in tables to be described below.

The NLQS database organization shown is intricately linked to the switched grammar architecture described in patent application.

Table Organization

There is a specific database for each Course. Each database includes three types of tables as follows: Master Table as illustrated in **Figure 7A**, Chapter Tables as illustrated in **Figure 7B** and Section Table as illustrated in **Figure 7C**.

As illustrated in **Fig. 7A,** the Master Table has six columns – Field Name **701A**, Data Type **702A**, Size **703A**, Null **704A**, Primary Key **705A** and Indexed **706A**. The Master Table has only two fields – Chapter Name **707A** and Section Name **708A**. Both CourseName and Section Name are commonly indexed.

The Chapter Table is illustrated in **Figure 7B**. The Chapter Table has six (6) columns – Field Name **720**, Data Type **721**, Size **722**, Null **723**, Primary Key **724** and Indexed **725**.

There are nine (9) rows of data – Chapter_ID **726**, Answer_ID **727**, Section Name **728**, Answer_Title **729**, PairedQuestion **730**, AnswerPath **731**, Creator **732**, Date of Creation **733** and Date of Modification **734**.

This Chapter Table contains eight (8) fields as illustrated in **Figure 7C**. Each of the eight (8) Fields **730** has a description 731 and stores data corresponding to:

AnswerID **732** – an integer that is automatically incremented for user convenience
Section_Name **733** – the name of the section to which the particular record belongs This field along with the AnswerID has to be made the primary key
Answer_Title **734** – A short description of the title
PairedQuestion **735** – Contains one or more combinations of questions for the related answers whose path is stored in the next column AnswerPath
AnswerPath **736** – contains the path of the text file, which contains the answer to the related questions stored in the previous column
Creator **737**– Name of Content Creator
Date_of_Creation **738** – Date on which content was created
Date of Modification **739** – Date on which content was changed or modified

The Section Table is illustrated in **Figure 7D**. The Section Table has six (6) columns – Field Name **740**, Data Type **741**, Size **742**, Null **743**, Primary Key **744** and Indexed **745**. There are seven (7) rows of data – Answer_ID **746**, Answer_Title **747**, PairedQuestion **748**, AnswerPath **749**, Creator **750**, Date of Creation **751** and Date of Modification **752**.

SQL Full-Text Search

The query components accept a full-text predicate or rowset-valued function from SQL Server, transform parts of the predicate into an internal format, and send it to the Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the result set that is then returned to the database processor

.

The SQL Server Relational Engine accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the *ft_search_condition* and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query.

The Full-Text Provider parses and validates *ft_search_condition*, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy *ft_search_condition*. The handling of this rowset is conceptually similar to the code used in support of the **OPENROWSET()** and **OPENQUERY()** rowset-valued functions.

Full-Text Catalogs

This is where full-text indexes reside. This is a file-system directory that is accessible only by Administrator and the Search Service. The full-text indexes are organized into full-text

catalogs, which are referenced by friendly names. Typically, the full-text index data for an entire database is placed into a single full-text catalog.

Full-Text Indexing Administration
The following steps are started by an administrator using GUIs or stored procedures (either on demand or as scheduled). First, enable the database for full-text search and then identify the tables and columns that are to be registered for full-text search. This information is stored in the SQL Server system tables. When a table is activated for full-text processing, a population start seed for that table is sent to indexing support. Next, request the initial population of the full-text index for a table. Actually, the granularity of population is a full-text catalog, so if more than one table has been linked to a full-text catalog, the result is the full population of the full-text indexes for all tables linked to that catalog.

The knowledge of the tables and rows that require indexing resides in SQL Server, so when indexing support receives a population request for a full-text catalog, it calls back into SQL Server to obtain data from all the columns in the table that have been marked for indexing. When this data arrives, it is passed to the index engine where it is broken into words. Noise words are removed and the remaining words are stored in the index. Full-text indexes are kept current by using a GUI that sets up a schedule for periodic refreshes of a full-text catalog. This GUI uses stored procedures from the SQL Server job scheduler. It also is possible to request a refresh at any time, either by means of a GUI or by direct use of a stored procedure.

If a table has a row-versioning (timestamp) column, repopulation can be handled more efficiently. At the time the population start seed for a table is constructed, the largest row-versioning value in the database is remembered. When an incremental population is requested, the SQL Server handler connects to the database and requests only rows where the row-versioning value is greater than the remembered value.
During an incremental population, if there is a full-text indexed table in a catalog that does not have a row-versioning column, then that table will be completely repopulated.

There are two cases where a complete re-population is performed, even though there is a row-versioning column on the table. These are:

1. In tables when the schema has changed.
2. In tables activated since the last population.

After populating tables with a row-versioning column, the remembered row-versioning value is updated. Since incremental repopulation on relatively static tables with timestamp columns can be completed faster than a complete repopulation, users can schedule repopulation on a more frequent basis. For a given database, users should take care not to mix index data from tables with timestamp columns with tables in the same full-text catalog, because these two groups of tables usually should be on separate repopulation schedules.

Additional tables and columns will be registered for full-text search, and full-text indexes will be generated for them. The full-text search capability may be removed from some tables and columns. Some full-text catalogs may be dropped. This step may be repeated several times.

Search Service and Search Engine –TheSearch Service is performed by the Search Engine. It performs two basic functions:

1. Indexing support accepts requests to populate the full-text index of a given table.
2. Querying support processes full-text searches.

The Search Engine processes full-text search queries. It determines which entries in the index meet the selection criteria. For each entry that meets the selection criteria, the value of the unique key column and a ranking value are returned.

As illustrated in **Fig 1000**, the Full-Text Query Process is as follows:

1. A query **1001** that uses one of the SQL full-text constructs is submitted to the SQL Relational Engine.
2. Queries containing either the **CONTAINS** or **FREETEXT** predicate are rewritten **1003** so that the rowset returned from the Full-Text Provider later will be automatically joined to the table that the predicate is acting upon. This rewrite is the mechanism used to ensure that these predicates are a seamless extension to SQL Server.
3. The Full-Text Provider is invoked, passing the following information:
   a. The *ft_search_condition*
   b. The friendly name of the full-text catalog where the full-text index of a table resides
   c. The locale ID to be used for language (for example, word breaking)
   d. The identities of the database, table, and column
   e. If the query is comprised of more than one full-text construct, the full-text provider is invoked separately for each construct.
4. The SQL Relational Engine **1002** does not examine the contents of the *ft_search_condition*. Instead, this is passed along to the full-text provider **1007**, which verifies the validity and then creates the appropriate internal representation of the full-text search condition.
5. The command is then passed to Querying Support **1011**.
6. Querying Support **1012** returns a rowset **1009** that contains the unique key column values for the rows that match the full-text search criteria. A rank value also is returned for each row.
7. The rowset is passed **1005** to the SQL Relational Engine **1002.** If processing either a **CONTAINSTABLE()** or **FREETEXTTABLE()** function, **RANK** values are returned; otherwise, the rank value is filtered out.
8. The rowset values are plugged into the query with values obtained from the relational database, and the result set **1015** is returned to the user.

The the recordset containing paired questions is passed again to the NLE. This step is the 2nd step of the 2-step algorithm as illustrated in **Fig. 8B**. Its carries out detailed linguistic analysis of the paired questions and in addition iteratively compares the NP of each record of the recordset with that of the NP of the user's question to identify the stored question that best matches the user's question. The criterion against which the best stored question is selected is the maximum number of NP, that is, the paired question that has the maximal number of NP is chosen to be the one that best matches the user's question.

As illustrated in **Fig. 5**, after the recordset is returned from the database, and the array of paired questions are stored as an array, this array is then passed to **14** of the NLE. The sequence of events that follow are similar to that used to extract the NP of the user's question. In this analysis, the noun phrases are extracted from each of the paired questions in sequence.

As illustrated in **Fig 5**, the entry point for the array of paired questions to the NLE is port **14** of the NLE. The processing steps that follow are via several sub-blocks labeled **9a** through **9e**. Each of these sub-blocks implement the several different linguistic functions performed by the NLE that are required to extract the noun phrases from the array of paired questions. The steps are as follows:

Initialize Grouper Resources Object and the Library **9a**, this block initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

Tokenizing of the words from the given text **9b** - in this block all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

The Tagging of all the tokens are implemented in **9c**. Here the Tagger object will tag all the tokenized words and the output is passed to Grouper object.

The Grouping of all tagged token to form NP list **9d** - in this block the Grouper object groups all the tagged token words and gives out as Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources 9e – in this block all the initialized resources are un-initialized and the resources are freed. Also de-allocates the memory allocated to store all Noun Phrases

Again referring to **Fig. 5**, at the end of this processing, the NP list is returned from the NLE to block **13** of **Fig. 5**. Specifically, it is passed to the block **15** – Get Best Answer ID. Before it extracts the identification (ID) for the single best answer, processing passes to Block **15a** where a comparison of the NP of user's question with NP of the Paired Questions extracted from database is made to find out the best suitable question present in the database that matches the user's question.

Block **15** then passes the Best Answer ID to block **16** of **Fig. 5.** Some preparatory steps are executed before processing takes place in **block 16**. These steps are as follows:

1. Creates a database process DLL class object
2. Gets the NP list from the question and builds the full-text SQL query using the NP list from the question and section name
3. Gets all the paired questions related to the user's questions from the database. If no records are fetched, then an error is returned. If only one record is returned, then the answer is returned using the answer's file path

from the fetched single record. Otherwise, the NLE is called to get the single best record.

As illustrated in **Fig. 4-5-2**, block **716a** of this diagram receives the "best record number". The file path for this "best record number" is then fetched in block **716b.** Next that particular file is opened in block **716c**. Also in the same block **716c**, the contents of the file are read and the contents passed to block **716d**.

In determining the file path, the following are the steps:

1. The record pointer is moved to the specified record number.
2. The path of the answer file is next retrieved
3. The content from the file is read
4. The answer is returned

Once the answer is returned, the file is sent to block **716d**, where it is compressed. The actual sequence of steps involved are as follows:

1. The size of the answer to be compressed is obtained
2. Sufficient memory is allocated to hold the compressed answer.
3. The compressed answer and the size of the original answer is returned to the client.

Interface between NLE and DBProcess
This part of the code contains functions, which interface processes resident in the NLE block and DBProcess blocks. The functions are illustrated in **Fig. 4-5-3. 800** implements functions that extracts the Noun Phrase (NP) list from the user's question. This part of the code is a function, which interacts with the NLE and gets the list of Noun Phrases in a sentence. This function is used to find out the Noun Phrases existed in the user query. **813** retrieves the NP list from the list of paired questions and stores these questions into an array. This part of the code is a function, which interacts with the NLE and gets the list of Noun phrases in a sentence. This function is used to find out the Noun phrases existed in the paired questions that are similar to the user question.

Next the function - Get Best Answer ID is implemented as illustrated **815**. This part of the code gets the best answerID to the user's query. This function first finds out the number of Noun phrases that are matching with the Noun phrases in the user's query. Then it selects the record, which contains the maximum number of matching Noun phrases.

Then a comparison of the number of NP in the User's question with that of NP list of paired questions is implemented **815a .** This part of the code is a function which is used by block **815a** of the code to compare the find out the number of matching Noun Phrases in the given user's query and a given paired question which is similar to the user asked question.

The ID corresponding to the best answer is then returned to the Get Best Answer ID **815,** which then returns it to the DBProcess block.

Natural Language Engine
Again referring to **Fig. 4-5-3**, this engine implements the word analysis or morphological analysis of words that make up the user's query, and phrase analysis of the phrases extracted

from the query. As illustrated in **Fig. 8A**, the functions that make up the morphological analysis include tokenizers **802A**, stemmers **804A** and morphological analyzers **806A**. The functions that comprise the phrase analysis include tokenizers, taggers and groupers. The tokenizer is an function that breaks up an sentence into list of tokens **803A**. In performing this function, the tokenizer goes through the input text and treats it as a series of tokens or useful meaningful units that are typically larger than individual characters, but smaller than phrases and sentences. These include words, separable parts of word and punctuation. Each token is associated with an offset and a length. The first phase of tokenization is segmentation, which extracts the individual tokens from the input text and keeps track of the offset where each token originated from in the input text. Next categories are associated with each token, base do its shape.

Following tokenization, the stemmer which might be of two forms – inflectional and derivational, analyzes the tokens for its stems **805A**. The inflectional stemmer recognizes affixes and returns the word which is the stem. The derivational stemmer on the other hand recognizes derivational affixes and returns the root word or words.

As illustrated in **Fig. 8**, phrase analysis is the next step that is performed after tokenization. The tokens are assigned a parts of speech tag by the tagger **804**, and the grouper **806** recognizes groups of words as phrases of a certain syntactic type. These syntactic types include for example the noun phrases. Specifically, the tagger is a parts of speech disambiguator which analyzes words in context. It's built-in morphological analyzer allows it to identify all possible parts of speech for each token. The output of the tagger is a string with each token tagged with a parts of speech. The final step inn the linguistic process is the grouping of words to form phrases. This function is performed by the grouper **806**, and is very dependent on the output of the tagger component.

The general linguistic functions implemented by the NLE are illustrated in **Fig 8**. The input text **801** is first tokenized **802** to produce a list of tokens **803.** Then the list of tokens is then tagged by the Tagger **804** to produce the different parts of speech **805** for each token. Next each parts of speech is grouped by the grouper **806** to produce the list of phrases -- for example, a list of noun phrases (NP). The key purpose of the NLE module is to refine the search for the best answer so that a single-best answer is provided for the user's question using linguistic processing.

The entry point of NLE **900** is made up of several components **9a** through **9e**. Each of these components implement the several different functions required in the NLE.

Initialize Grouper Resources Object and the Library **900** – initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

Tokenizing of the words from the given text **909B** – here all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

The Tagging of all the tokens **909C**. Here the Tagger object tags all the tokenized words and the output is passed to Grouper object.

The Grouping of all tagged token to form NP list **909D** is implemented here so that the Grouper object groups all the tagged token words and outputs the Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources, **9E** in this component all the initialized resources are un-initialized. These include Token Resources **909EA**, Tagger Resources **909EA**, Grouper Resources **909EC**. After initialization, the and the resources are freed. The memory that was used to store all Noun Phrases are also de-allocated.

~~5.3~~ SERVER-SIDE DESCRIPTION

Introduction
The flow diagram of the algorithm for the Natural Language Query System as implemented at the server is illustrated in **Fig. 1100A** through **Fig. 1100C**. This algorithm is a 2-step algorithm. The 1st step as illustrated in **Fig. 1100A** is as follows: the user's query after being recognized **1101**, is sent to the NLE **1107**, and the DBProcess **1102**. At the NLE, the text string undergoes morphological linguistic processing **1108**: the string corresponding to the query is tokenized, the tags are tagged and the tagged tokens are grouped. Next the noun phrases (NP) of the string is stored **1109**, and also copied to the DBProcess **1110.**

As illustrated in **Fig. 1100A**, the string corresponding to the user's query which was sent to the DBProcess **1102**, is used together with the NP received from the NLE to construct the SQL Query **1103**. Next the SQL query is executed **1104**, and the recordset received from the full-text search **1105** is then sent back to the NLE in the form of an array **1106**.

Referring to **Fig. 1100B**, the 2nd step of the 2-step algorithm begins with the linguistic processing of each of the stored questions returned by the full-text search process. Processing of these stored questions continues in the NLE as follows: each question in the array of questions corresponding to the recordset returned by the SQL full-text search also undergoes morphological linguistic processing **1111**: the string corresponding to the stored question is tokenized, the tags are tagged and the tagged tokens are grouped. Next the noun phrases of the string is computed and stored **1112.** This process continues iteratively **1113, and 1118,1111, 1112, 1113** so that the NP for each stored question is computed and stored. Once NP for all the stored questions of the array are computed, a comparison is made between each stored question and the user's query based on the magnitude of the NP **1114**. This process is iterative **1114**, **1115**, **1116**, **1119** so that the comparison of the NP for each stored question with that of the NP of the user's query is completed. When there are no more stored questions in the array to be processed **1117**, the stored question that has the maximum NP relative to the user's query, is identified **1117A** as the stored question which best matches the user's query.

As illustrated in **Fig. 1100C,** the identity of the matching stored question **1120** is completed. Next the file path corresponding to the answer of the identified matching question is extracted **1121**. Processing continues so that the answer is extracted from the file path **1122** and finally the answer is compressed and sent to the client **1123.**

The discussion that follows describes in more detail each of the process steps of the NLQS algorithm in each respective sub-system.

Server System
The server-side of the NLQS system as illustrated in **Fig. 5** is comprised of several key processing modules. These include the Communication module – CommunicationServerISAPI **500**, the database processor DBProcess **501**, the Natural language engine (NLE) **9, 14**, and the interface between the NLE and the DBProcess **500B**. The Communication module – CommunicationServerISAPI **500** includes the server-side

speech recognition engine and the communication interfaces required between the client and server.

As illustrated in **Fig. 5**, the server-side logic of the Natural Language Query System as consists of the following two dynamic link library components: CommunicationServerISAPI **500** and the DBProcess **501**. The CommunicationServerIASPI is comprised of 3 modules: the Server-side Speech Recognition Engine **501A**; the Interface module between the Natural Language Engine and the DBProcess. **500B**; and the Natural Language Engine **500C.**

The DB Process **501**, is a module whose primary function is to connect to a SQL database and to execute the SQL query. In addition, it interfaces with logic that fetches the correct answer from a file path once this answer is passed to it from the Natural Language Engine **500C**.

Speech Recognition: Server-Side
The speech recognition engine **500A** is a set of distributed components of the speech recognition engine that reside at the server-side. Referring to **Fig. 4-5-1**, the signal flow for speech recognition at the server-side is as follows:

Within **601**, the received binary MFCC vector byte stream corresponding to the speech signal's acoustic features extracted at the client and sent over the communication channel is received. The MFCC acoustic vectors are decoded from the encoded HTTP byte stream as follows: Since the MFCC vectors contains embedded NULL characters, they cannot be transferred to the server as such using http protocol. Thus the MFCC vectors are encoded at the client-side before transmission in such a way that all the data converted into a stream of bytes without embedded NULL characters in the data. At the very end of the byte stream a single NULL character is introduced.

Dictionary Preparation & Grammar Files
Referring to **Fig. 4-5-1**, within block **605**, the options selected by the user: Course, Chapter and/or section are received. These selected options define the scope – i.e. grammars and dictionaries hat will be dynamically loaded to speech recognition engine for Viterbi decoding. For the speech recognition to be implemented, there has to be a both grammar and dictionary files. The Grammar file supplies all the sentences that are to be recognized will be provided, while the Dictionary file provides phonemes (the information of how to pronounce a word) of each word in the grammar file. If all the sentences for a given environment that can be recognized are contained in a single grammar file then the recognition accuracy will be deteriorated and the speed of loading the grammar and dictionary files would impair the speech recognition process. To avoid these problems, specific grammars are loaded or actively made to be the current grammar according the Course, Chapter and/or Section selected. Thus the grammar and dictionary files are loaded dynamically according to the given Course, Chapter and/or Section as dictated by the user. The second block **2** implements the initialization of the Speech Recognition engine. The MFCC vectors along with the grammar filename and the dictionary file names are introduced to this block to initialize of the speech decoder.

As illustrated in **Fig. 4-5-1**, the initialization process consists of the following processes:

Loading the SRE library **602a**. This then allows the creation of an External Source **602b** using MFCC vectors received. It allocates the memory required to hold the recognition objects **602c**, and also creates and initializes the objects **602d** that are required for the recognition such as Source, Coder, Recognizer and Results Loading of the Dictionary **602e** and Hidden Markov Models (HMMs) **602f**; Loading of the Grammar file as illustrated in **602g**.

Speech Recognition **603** is the next step as illustrated in **Fig. 4-5-1**. Using the functions created in the External Source, this block reads MFCC vectors one at a time from the External Source **603a**, and processes them in block **603b** to realize the words in the speech pattern that are symbolized by the MFCC vectors captured at the client.

Once the user's speech is recognized, the flow of the SRE passes to Un-initialize SRE **604** where the speech engine is un-initialized as illustrated. In this block all the objects created in the initialization block are deleted **604a**, and memory allocated in the initialization block during the initialization phase are removed **604b**.

Database Processor – DBProcess
Construction of the SQL Query as illustrated in **Fig. 4-5-4,** Construct SQL Query using the **SELECT** and **CONTAINS** predicate **950**, is the module that constructs the SQL query used for retrieving the best suitable question stored in the database. The table name is then concatenated with the constructed **SELECT** statement **951**. Next the number of words present in each Noun Phrase of Question asked by the user is first calculated **952**. Then as much memory is allocated **953** for all the words present in the NP. Next the word List present in the NP **954** is obtained. Next, these words are concatenated to the SQL Query separated with the **NEAR ( )** keyword **955**. Next, the **AND** keyword is concatenated to the SQL Query after each NP **956.** Finally the memory resource is freed so as to allocate memory to store the words received from NP for the next iteration **957**.

As illustrated in **Fig. 5**, using all the words present in the Noun Phrase along with Database name (Course), Table name (Chapter and/or Section) chosen by the user at client side, the SQL Query is constructed **10**. It also uses full-text predicate **CONTAINS** and other predicates such as **NEAR( )** and the **AND** operator while preparing the SQL Query. This customized SQL Query is then passed to the DBProcess **12** so that the paired questions can be retrieved from the NLQS database.

Connection to the SQL Server – As illustrated in **Fig. 4-5-2**, **711** implements the Connection to the Database. The connection sequence and the subsequent retrieved recordset is implemented in the following steps:

1. Server and database names are assigned to the DBProcess member variable **711A**
2. Connect the SQL Server database, **711C**
3. The SQL Query is received, **712A**
4. Execute the SQL query, **712B**
5. Extract the total number of records retrieved by the query
6. Allocate the memory to store the total number of paired questions
7. Store the entire number of paired questions into an array

In connecting to the SQL Server itself and to the database that is specified in the code of the DBProcess, the following 3 steps are executed:

1. The connection object is prepared
2. The connection string is prepared using the database name, user name, password, and the server name, **711A**
3. Then a connection is made to the SQL Server database, **711C**

Now referring back to **Fig. 5**, the next step is the Execution of the SQL Query **12B**. To execute the SQL Query, the SQL Query constructed in Block **10** is passed to **712a** of **Fig. 4-5-2**. After assembling there, it is passed to block **712b** where is executed.

Then there is the creation of code in this same block **712b** that creates a recordset object so that it accommodate the recordset after it is returned from the NLQS database.

NLQS Database Table Organization

Figure 7 illustrates structure of tables used in a typical NLQS database. The NLQS database that is used part of the query system required in training environment consists typically of a Course 7**01**, which is made of several chapters **702**, **703**, **704**. Each of these chapters can have one or more Sections **705**, **706**, **707** as shown for Chapter 1. A similar structure can exist for Chapter 2, Chapter 3 … Chapter N.

Each section has a set of one or more question answer pairs **708**, **709**, **710** stored in tables to be described below.

The NLQS database organization shown is intricately linked to the switched grammar architecture described in patent application.

Table Organization

There is a specific database for each Course. Each database includes three types of tables as follows: Master Table as illustrated in **Figure 7A**, Chapter Tables as illustrated in **Figure 7B** and Section Table as illustrated in **Figure 7C**.

As illustrated in **Fig. 7A,** the Master Table has six columns – Field Name **701A**, Data Type **702A**, Size **703A**, Null **704A**, Primary Key **705A** and Indexed **706A**. The Master Table has only two fields – Chapter Name **707A** and Section Name **708A**. Both CourseName and Section Name are commonly indexed.

The Chapter Table is illustrated in **Figure 7B**. The Chapter Table has six (6) columns – Field Name **720**, Data Type **721**, Size **722**, Null **723**, Primary Key **724** and Indexed **725**. There are nine (9) rows of data – Chapter_ID **726**, Answer_ID **727**, Section Name **728**, Answer_Title **729**, PairedQuestion **730**, AnswerPath **731**, Creator **732**, Date of Creation **733** and Date of Modification **734**.

This Chapter Table contains eight (8) fields as illustrated in **Figure 7C**. Each of the eight (8) Fields **730** has a description 731 and stores data corresponding to:

AnswerID **732** – an integer that is automatically incremented for user convenience
Section_Name **733** – the name of the section to which the particular record belongs
This field along with the AnswerID has to be made the primary key

Answer_Title **734** – A short description of the title
PairedQuestion **735** – Contains one or more combinations of questions for the related answers whose path is stored in the next column AnswerPath
AnswerPath **736** – contains the path of the text file, which contains the answer to the related questions stored in the previous column
Creator **737**– Name of Content Creator
Date_of_Creation **738** – Date on which content was created
Date of Modification **739** – Date on which content was changed or modified

The Section Table is illustrated in **Figure 7D**. The Section Table has six (6) columns – Field Name **740**, Data Type **741**, Size **742**, Null **743**, Primary Key **744** and Indexed **745**. There are seven (7) rows of data – Answer_ID **746**, Answer_Title **747**, PairedQuestion **748**, AnswerPath **749**, Creator **750**, Date of Creation **751** and Date of Modification **752**.

SQL Full-Text Search
The query components accept a full-text predicate or rowset-valued function from SQL Server, transform parts of the predicate into an internal format, and send it to the Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the result set that is then returned to the database processor
.
The SQL Server Relational Engine accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the *ft_search_condition* and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query.

The Full-Text Provider parses and validates *ft_search_condition*, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy *ft_search_condition*. The handling of this rowset is conceptually similar to the code used in support of the **OPENROWSET()** and **OPENQUERY()** rowset-valued functions.

Full-Text Catalogs
This is where full-text indexes reside. This is a file-system directory that is accessible only by Administrator and the Search Service. The full-text indexes are organized into full-text catalogs, which are referenced by friendly names. Typically, the full-text index data for an entire database is placed into a single full-text catalog.

Full-Text Indexing Administration
The following steps are started by an administrator using GUIs or stored procedures (either on demand or as scheduled). First, enable the database for full-text search and then identify the tables and columns that are to be registered for full-text search. This information is stored in the SQL Server system tables. When a table is activated for full-text processing, a population start seed for that table is sent to indexing support. Next, request the initial population of the full-text index for a table. Actually, the granularity of population is a full-

text catalog, so if more than one table has been linked to a full-text catalog, the result is the full population of the full-text indexes for all tables linked to that catalog.

The knowledge of the tables and rows that require indexing resides in SQL Server, so when indexing support receives a population request for a full-text catalog, it calls back into SQL Server to obtain data from all the columns in the table that have been marked for indexing. When this data arrives, it is passed to the index engine where it is broken into words. Noise words are removed and the remaining words are stored in the index. Full-text indexes are kept current by using a GUI that sets up a schedule for periodic refreshes of a full-text catalog. This GUI uses stored procedures from the SQL Server job scheduler. It also is possible to request a refresh at any time, either by means of a GUI or by direct use of a stored procedure.

If a table has a row-versioning (timestamp) column, repopulation can be handled more efficiently. At the time the population start seed for a table is constructed, the largest row-versioning value in the database is remembered. When an incremental population is requested, the SQL Server handler connects to the database and requests only rows where the row-versioning value is greater than the remembered value.
During an incremental population, if there is a full-text indexed table in a catalog that does not have a row-versioning column, then that table will be completely repopulated.

There are two cases where a complete re-population is performed, even though there is a row-versioning column on the table. These are:

1. In tables when the schema has changed.
2. In tables activated since the last population.

After populating tables with a row-versioning column, the remembered row-versioning value is updated. Since incremental repopulation on relatively static tables with timestamp columns can be completed faster than a complete repopulation, users can schedule repopulation on a more frequent basis. For a given database, users should take care not to mix index data from tables with timestamp columns with tables in the same full-text catalog, because these two groups of tables usually should be on separate repopulation schedules.

Additional tables and columns will be registered for full-text search, and full-text indexes will be generated for them. The full-text search capability may be removed from some tables and columns. Some full-text catalogs may be dropped. This step may be repeated several times.

Search Service and Search Engine –TheSearch Service is performed by the Search Engine. It performs two basic functions:

1. Indexing support accepts requests to populate the full-text index of a given table.
2. Querying support processes full-text searches.

The Search Engine processes full-text search queries. It determines which entries in the index meet the selection criteria. For each entry that meets the selection criteria, the value of the unique key column and a ranking value are returned.

As illustrated in **Fig 1000**, the Full-Text Query Process is as follows:

1. A query **1001** that uses one of the SQL full-text constructs is submitted to the SQL Relational Engine.
2. Queries containing either the **CONTAINS** or **FREETEXT** predicate are rewritten **1003** so that the rowset returned from the Full-Text Provider later will be automatically joined to the table that the predicate is acting upon. This rewrite is the mechanism used to ensure that these predicates are a seamless extension to SQL Server.
3. The Full-Text Provider is invoked, passing the following information:
   a. The *ft_search_condition*
   b. The friendly name of the full-text catalog where the full-text index of a table resides
   c. The locale ID to be used for language (for example, word breaking)
   d. The identities of the database, table, and column
   e. If the query is comprised of more than one full-text construct, the full-text provider is invoked separately for each construct.
4. The SQL Relational Engine **1002** does not examine the contents of the *ft_search_condition*. Instead, this is passed along to the full-text provider **1007**, which verifies the validity and then creates the appropriate internal representation of the full-text search condition.
5. The command is then passed to Querying Support **1011**.
6. Querying Support **1012** returns a rowset **1009** that contains the unique key column values for the rows that match the full-text search criteria. A rank value also is returned for each row.
7. The rowset is passed **1005** to the SQL Relational Engine **1002.** If processing either a **CONTAINSTABLE()** or **FREETEXTTABLE()** function, **RANK** values are returned; otherwise, the rank value is filtered out.
8. The rowset values are plugged into the query with values obtained from the relational database, and the result set **1015** is returned to the user.

The the recordset containing paired questions is passed again to the NLE. This step is the 2nd step of the 2-step algorithm as illustrated in **Fig. 8B**. Its carries out detailed linguistic analysis of the paired questions and in addition iteratively compares the NP of each record of the recordset with that of the NP of the user's question to identify the stored question that best matches the user's question. The criterion against which the best stored question is selected is the maximum number of NP, that is, the paired question that has the maximal number of NP is chosen to be the one that best matches the user's question.

As illustrated in **Fig. 5**, after the recordset is returned from the database, and the array of paired questions are stored as an array, this array is then passed to **14** of the NLE. The sequence of events that follow are similar to that used to extract the NP of the user's question. In this analysis, the noun phrases are extracted from each of the paired questions in sequence.

As illustrated in **Fig 5**, the entry point for the array of paired questions to the NLE is port **14** of the NLE. The processing steps that follow are via several sub-blocks labeled **9a** through **9e**. Each of these sub-blocks implement the several different linguistic functions performed by the NLE that are required to extract the noun phrases from the array of paired questions. The steps are as follows:

Initialize Grouper Resources Object and the Library **9a**, this block initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

Tokenizing of the words from the given text **9b** - in this block all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

The Tagging of all the tokens are implemented in **9c**. Here the Tagger object will tag all the tokenized words and the output is passed to Grouper object.

The Grouping of all tagged token to form NP list **9d** - in this block the Grouper object groups all the tagged token words and gives out as Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources 9e – in this block all the initialized resources are un-initialized and the resources are freed. Also de-allocates the memory allocated to store all Noun Phrases

Again referring to **Fig. 5**, at the end of this processing, the NP list is returned from the NLE to block **13** of **Fig. 5**. Specifically, it is passed to the block **15** – Get Best Answer ID. Before it extracts the identification (ID) for the single best answer, processing passes to Block **15a** where a comparison of the NP of user's question with NP of the Paired Questions extracted from database is made to find out the best suitable question present in the database that matches the user's question.

Block **15** then passes the Best Answer ID to block **16** of **Fig. 5.** Some preparatory steps are executed before processing takes place in **block 16**. These steps are as follows:

1. Creates a database process DLL class object
2. Gets the NP list from the question and builds the full-text SQL query using the NP list from the question and section name
3. Gets all the paired questions related to the user's questions from the database. If no records are fetched, then an error is returned. If only one record is returned, then the answer is returned using the answer's file path from the fetched single record. Otherwise, the NLE is called to get the single best record.

As illustrated in **Fig. 4-5-2**, block **716a** of this diagram receives the "best record number". The file path for this "best record number" is then fetched in block **716b.** Next that particular file is opened in block **716c**. Also in the same block **716c**, the contents of the file are read and the contents passed to block **716d**.

In determining the file path, the following are the steps:

1. The record pointer is moved to the specified record number.
2. The path of the answer file is next retrieved

3. The content from the file is read
4. The answer is returned

Once the answer is returned, the file is sent to block **716d**, where it is compressed. The actual sequence of steps involved are as follows:

1. The size of the answer to be compressed is obtained
2. Sufficient memory is allocated to hold the compressed answer.
3. The compressed answer and the size of the original answer is returned to the client.

Interface between NLE and DBProcess
This part of the code contains functions, which interface processes resident in the NLE block and DBProcess blocks. The functions are illustrated in **Fig. 4-5-3. 800** implements functions that extracts the Noun Phrase (NP) list from the user's question. This part of the code is a function, which interacts with the NLE and gets the list of Noun Phrases in a sentence. This function is used to find out the Noun Phrases existed in the user query. **813** retrieves the NP list from the list of paired questions and stores these questions into an array. This part of the code is a function, which interacts with the NLE and gets the list of Noun phrases in a sentence. This function is used to find out the Noun phrases existed in the paired questions that are similar to the user question.

Next the function - Get Best Answer ID is implemented as illustrated **815**. This part of the code gets the best answerID to the user's query. This function first finds out the number of Noun phrases that are matching with the Noun phrases in the user's query. Then it selects the record, which contains the maximum number of matching Noun phrases.

Then a comparison of the number of NP in the User's question with that of NP list of paired questions is implemented **815a .** This part of the code is a function which is used by block **815a** of the code to compare the find out the number of matching Noun Phrases in the given user's query and a given paired question which is similar to the user asked question.

The ID corresponding to the best answer is then returned to the Get Best Answer ID **815,** which then returns it to the DBProcess block.

Natural Language Engine
Again referring to **Fig. 4-5-3**, this engine implements the word analysis or morphological analysis of words that make up the user's query, and phrase analysis of the phrases extracted from the query. As illustrated in **Fig. 8A**, the functions that make up the morphological analysis include tokenizers **802A**, stemmers **804A** and morphological analyzers **806A**. The functions that comprise the phrase analysis include tokenizers, taggers and groupers. The tokenizer is an function that breaks up an sentence into list of tokens **803A**. In performing this function, the tokenizer goes through the input text and treats it as a series of tokens or useful meaningful units that are typically larger than individual characters, but smaller than phrases and sentences. These include words, separable parts of word and punctuation. Each token is associated with an offset and a length. The first phase of tokenization is segmentation, which extracts the individual tokens from the input text and keeps track of the offset where each token originated from in the input text. Next categories are associated with each token, base do its shape.

Following tokenization, the stemmer which might be of two forms – inflectional and derivational, analyzes the tokens for its stems **805A**. The inflectional stemmer recognizes affixes and returns the word which is the stem. The derivational stemmer on the other hand recognizes derivational affixes and returns the root word or words.

As illustrated in **Fig. 8**, phrase analysis is the next step that is performed after tokenization. The tokens are assigned a parts of speech tag by the tagger **804**, and the grouper **806** recognizes groups of words as phrases of a certain syntactic type. These syntactic types include for example the noun phrases. Specifically, the tagger is a parts of speech disambiguator which analyzes words in context. It's built-in morphological analyzer allows it to identify all possible parts of speech for each token. The output of the tagger is a string with each token tagged with a parts of speech. The final step inn the linguistic process is the grouping of words to form phrases. This function is performed by the grouper **806**, and is very dependent on the output of the tagger component.

The general linguistic functions implemented by the NLE are illustrated in **Fig 8**. The input text **801** is first tokenized **802** to produce a list of tokens **803.** Then the list of tokens is then tagged by the Tagger **804** to produce the different parts of speech **805** for each token. Next each parts of speech is grouped by the grouper **806** to produce the list of phrases -- for example, a list of noun phrases (NP). The key purpose of the NLE module is to refine the search for the best answer so that a single-best answer is provided for the user's question using linguistic processing.

The entry point of NLE **900** is made up of several components **9a** through **9e**. Each of these components implement the several different functions required in the NLE.

Initialize Grouper Resources Object and the Library **900** – initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

Tokenizing of the words from the given text **909B** – here all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

The Tagging of all the tokens **909C**. Here the Tagger object tags all the tokenized words and the output is passed to Grouper object.

The Grouping of all tagged token to form NP list **909D** is implemented here so that the Grouper object groups all the tagged token words and outputs the Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources, **9E** in this component all the initialized resources are un-initialized. These include Token Resources **909EA**, Tagger Resources **909EA**, Grouper Resources **909EC**. After initialization, the and

the resources are freed. The memory that was used to store all Noun Phrases are also de-allocated.

Client System 150

The architecture of client-side system 150 of Natural Language Query System 100 is illustrated in greater detai in **Fig. 2**. Referring to **Fig. 2**, the three main processes effectuated by Client System 150 are illustrated as follows: Initialization process **200A** consisting of SRE **201,** Communication **202** and MS Agent **203** routines; an iterative process **200B** consisting of two sub-routines: a) Receive User Speech **208** – made up of SRE **204 and Communication 205;** and b) Receive Answer from Server **207** and an un-initialization process **200B**. Finally, un-initialization process 200C is made up of three sub-routines: SRE **212,** Communication **213**, and MS Agent **214.**

Deleted: processes

Deleted: processes

Deleted: The

Deleted: processes

Each of the above three processes are described in detail in the following paragraphs. It will be appreciated by those skilled in the art that the particular implementation for such processes and routines will vary from client platform to platform, so that in some environments such processes may be embodied in hard coded routines executed by a dedicated DSP, while in others they may be embodied as software routines executed by a shared host processor, and in still others a combination of the two may be used.

Initialization at Client System 150

The initialization of the Client System 150 is illustrated in **Fig. 2-2** and is comprised of initializing 3 processes: the client-side Speech Recognition Engine **220A**, the MS Agent **220B** and the Communication processes **220C**.

Deleted: client side

Deleted: as

Initialization of the Speech Recognition Engine

In block **1** the SRE COM Library is initialized, and memory **220** is allocated to hold the objects of Source and Coder. Source and Coder objects **221**are created. Loading of configuration file **221A** from the configuration file **221B** also takes place at the same time that the SRE Library is initialized. In the configuration file **221B**, the type of the input of Coder, the type of the output of the Coder are declared.

Next, the Speech and Silence components of an utterance are calibrated **222**. To calibrate the speech and silence components, the user articulates a sentence that is displayed in a text box on the screen. Then the SRE library estimates the noise and other parameters required to find the silence and the speech.

Initialization of the MS Agent

Initialization is described by block **220B** as illustrated in **Fig. 2-2**. This initialization consists of the following steps:

1. Initialize COM library **223**. This part of the code initializes the COM library, which is required to use the ActiveX Controls.
2. Create instance of Agent Server **224** - this part of the code creates an instance of Agent ActiveX control.
3. Loading of the MS Agent **225** - this part of the code loads MS Agent character from the specified file **225A**.

Revised: 9/13/99

4. Get Character Interface **226** - this part of the code gets the interface for the specified character.
5. Add Commands to Agent Character Option **227** - this part of the code adds commands to Agent Properties sheet, which can be accessed by clicking on the icon that appears in the system tray, when the character is loaded e.g: Speak, TTS Properties.
6. Show the Agent Character **228** - this part of the code displays the character on the screen
7. AgentNotifySink - to handle events. This part of the code creates AgentNotifySink object **229**, registers it **230** and Get Agent Properties interface **231**. The property sheet to the Agent character is done in **232**.
8. Do Character Animations **233** - This part of the code plays specified character animations to welcome the user to the NLQS.

The above then constitutes the entire sequence required to initialize the MS Agent.

Initialization of the Communication Process
The initialization of the Communication Process **220C** is also illustrated in **Fig. 2-2**. Referring to **Fig. 2-2**, this initialization consists of the following components: Open Internet Connection **234** - this part of the code opens Internet Connection and sets the parameter for the connection. Then the Set Callback Status **235** sets the callback status so as to inform the user of the status of connection. Finally Start New HTTP Internet Session **236** starts a new Internet session.

Iterative Processing
As illustrated in **Fig. 3**, once the initialization is complete, an iterative process is launched when the user presses the Start Button to initiate a query. Referring to **Fig. 3**, the iterative process consists of two sub-processes: **Receive User Speech 240** and the **Receive User Answer 243.** The **Receive User Speech 240** receives speech from the user, while the **Receive User Answer 243** receives the answer to the user's question in the form of text from the server so that it can be converted to speech for the user by the text-to-speech engine.

Receive User Speech – As illustrated in **Fig. 3**, the SRE **241** and the Communication **242** processes are the two (2) processes involved in receiving the user's utterance. The coder **248** is prepared so that the coder object receives speech data from the source object. Next the Start Source **249** is initiated. This part of the code initiates the data retrieval using the source Object which will in turn be given to the Coder object. Next MFCC vectors **250** are extracted from the Speech utterance until silence is detected.

Communication – the communication module **242** is used to implement the transport of data from the client to the server over the Internet. The data consists of encoded MFCC vectors that will be used at then server-side of the Speech Recognition engine to complete the speech recognition decoding. The sequence of the communication is as follows:

1. **OpenHTTPRequest 251** - this part of the code first converts MFCC vectors to a stream of bytes and processes the bytes so that it is compatible with the HTTP protocol.

Revised: 9/13/99

2. **Encode the MFCC Byte Stream 251** - this part of the code encodes MFCC vectors, so that it can be sent to the server via http. (Regarding the encoding more information is given in the server side document)
3. **Send the data 252** - this part of the code sends MFCC vectors to the server using the Internet connection and the http protocol.
4. **Wait for the Server Response 253 -** this part of the code sends will be in the loop till the response from the server arrives

Receive Answer from Server 243 is comprised of the following modules as shown in **Fig. 3**.: the MS Agent **244**, Text-to-Speech Engine **245** and the Communication modules **246**. All three modules interact to receive the answer from the server. As illustrated in **Fig. 3**, the communication process consists of three processes: the Receive the Best Answer **258** receives the best answer over the HTTP communication channel. The answer is de-compressed in **259** and then the answer is passed to the MS Agent by **260**. MS Agent **254** receives the answer from Communication process **246**. MS Agent **255** then articulates the answer using the text-to-speech engine **257**. The text to speech engine uses the natural language voice data file **256** that is associated with it.

Uninitialization

The un-initialization is illustrated in **Fig. 4**. Three functions are un-initialized – the SRE **270**, Communication, **271** and the MS Agent **272**. To un-initialize the SRE, memory that was allocated in the initialization phase is de-allocated **273** and objects created during initialization phase are deleted **274.** As illustrated in **Fig. 4**, to un-initialize the Communication **271**, and the Internet connection previously established with the server is closed **275**. Next the Internet session created at the time of initialization id closed **276.** For the un-initialization of the MS Agent **272,**as illustrated in **Fig. 4**, the Commands Interface is first released **277**. This releases the commands added to property sheet during loading of character. Next the Character Interface is released **278** and the Agent is unloaded **279**. The Sink Object Interface is then released **280** followed by the release of the Property Sheet Interface **281**. The Agent Notify Sink **282** is then un-registered the Agent and finally the Agent Interface **283** is released which releases all the resources allocated during initialization.

Revised: 9/13/99

Express Mail Label No. US......................

DISTRIBUTED SPEECH-BASED INTERACTIVE SYSTEM AND METHOD THEREFOR

FIELD OF THE INVENTION

The invention relates to a system and an interactive method for responding to speech based user inputs and queries presented over a distributed network such as the Internet or local intranet. This interactive system when implemented over the world-wide web services (WWW) of the Internet, functions so that a client or user can ask a question in a natural language such as English, French, German, Spanish or Japanese and receive the appropriate answer at his or her computer or accessory also in his or her native natural language. The system has particular applicability to such applications as remote learning, e-commerce, technical e-support services, internet searching, etc.

BACKGROUND OF THE INVENTION

The Internet, and in particular, the World-Wide Web (WWW), is growing in popularity and usage for both commercial and recreational purposes, and this trend is expected to continue. This phenomenon is being driven, in part, by the increasing and widespread use of personal computer systems and the availability of low cost internet access. The emergence of inexpensive internet access devices and high speed access techniques such as ADSL, cable modems, satellite modems, and the like, are expected to further accelerate the mass usage of the WWW.

Accodingly, it is expected that the number of entities offering services, products, etc., over the WWW will increase dramatically over the coming years. Until now, however, the internet "experience" for users has been limited mostly to non-voice based input/output devices, such as keyboards, intelligent electronic pads, mice, trackballs, printers, monitors, etc. This presents somewhat of a bottleneck for interacting over the WWW for a variety of reasons.

First, there is the issue of familiarity. Many kinds of applications lend themselves much more naturally and fluently to a voice-based environment. For instance, most people shopping for audio recordings are very comfortable with asking a live sales clerk in a record

store for information on titles by a particular author, where they can be found in the store, etc. While it is often possible to browse and search on one's own to locate items of interest, it is usually easier and more efficient to get some form of human assistance first, and, with few exceptions, this request for assistance is presented in the form of a oral query. In addition, many persons cannot or will not, because of physical or psychological barriers, use any of the aforementioned conventional I/O devices. For example, many older persons cannot easily read the text presented on WWW pages, or understand the layout/hierarchy of menus, or manipulate a mouse to make finely coordinated movements to indicate their selections. Many others are intimidated by the look and complexity of computer systems, WWW pages, etc., and therefore do not attempt to use online services for this reason as well.

Thus, applications which can mimic normal human interactions are likely to be preferred by potential on-line shoppers and persons looking for information over the WWW. It is also expected that the use of voice-based systems will increase the universe of persons willing to engage in e-commerce, e-learning, etc. To date, however, there are very few systems, if any, which permit this type of interaction, and, if they do, it is very limited. For example, various commercial programs sold by IBM (Via Voice) and Kurzweil (Dragon) permit some user control of the interface (opening, closing files) and searching (by using previously trained URLs) but they do not present a flexible solution that can be used by a number of users across multiple cultures and without time consuming voice training.

Another issue presented by the lack of voice-based systems is efficiency. Many companies are now offering technical support over the internet, and some even offer live operator assistance for such queries. While this is very advantageous (for the reasons mentioned above) it is also extremely costly and inefficient, because a real person must be employed to handle such queries. This presents a practical limit that results in long wait times for responses or high labor overheads. In general, a service presented over the WWW is far more desirable if it is "scalable," or, in other words, able to handle an increasing amount of user traffic with little if any perceived delay or troubles by a prospective user.

In a similar context, while remote learning has become an increasingly popular option for many students, it is practically impossible for an instructor to be able to field questions from more than one person at a time. Even then, such interaction usually takes place for only a limited period of time because of other instructor time constraints. To date, however, there is no practical way for students to continue a human-like question and

answer type dialog after the learning session is over, or without the presence of the instructor to personally address such queries.

Conversely, another aspect of emulating a human-like dialog involves the use of oral feedback. In other words, many persons prefer to receive answers and information in audible form. While a form of this functionality is used by some websites to communicate information to visitors, it is not performed in a real-time, interactive question-answer dialog fashion so its effectiveness and usefulness is limited.

Yet another area that could benefit from speech-based interaction involves so-called "search" engines used by Internet users to locate information of interest at web sites, such as the those available at Yahoo.com, metacrawler.com, Excite.com, etc. These tools permit the user to form a search query using either combinations of keywords or metacategories to search through a web page database containing text indices associated with one or more distinct web pages. After processing the user's request, therefore, the search engine returns a number of hits which correspond, generally, to URL pointers and text excerpts from the web pages that represent the closest match made by such search engine for the particular user query based on the search processing logic used by search engine. The structure and operation of such prior art search engines, including the mechanism by which they build the web page database, and parse the search query, are well-known in the art. To date, applicant is unaware of any such search engine that can easily and reliably search and retrieve information based on speech input from a user.

There are a number of reasons why the above environments (e-commerce, e-support, remote learning, internet searching, etc.) do not utilize speech-based interfaces, despite the many benefits that would otherwise flow from such capability.

First, there is obviously a requirement that the output of the speech recognizer be as accurate as possible. One of the more reliable approaches to speech recognition used at this time is based on the Hidden Markov Model (HMM) – a model used to mathematically describe any time series. Because speech is considered to have an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of *states*, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called *transition probabilities*. In a particular state an outcome or *observation* can be generated, according to the associated

probability distribution. This finite state machine changes state once every time unit, and each time t such that a state j is entered, a spectral parameter vector O_t is generated with probability density $B_j(O_t)$. It is only the outcome, not the state visible to an external observer and therefore states are ``hidden" to the outside; hence the name Hidden Markov Model. [*Here we should make a reference to some contemporary patents, articles that describe HMM*]

While the HMM approach yields very good results, contemporary variations of this technique cannot guarantee an accuracy requirement of almost 100%, as will be required for WWW applications for all possible all user and environment conditions. Thus, although speech recognition technology has been available for several years, and has improved significantly, the technical requirements have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily. [*Note: we should explain generally what natural language process is, and some examples of its use in the art*]

Second, most of the very reliable voice recognition systems are speaker-dependent, requiring that the interface be "trained" with the user's voice, which takes a lot of time, and is thus very undesirable from the perspective of a WWW environment, where a user may interact only a few times with a particular website. Furthermore, speaker-dependent systems usually require a large user dictionary (one for each unique user) which reduces the speed of recognition. This makes it much harder to implement a real-time dialog interface with satisfactory response capability (i.e., something that mirrors normal conversation – on the order of 3 – 5 seconds is probably ideal). [*Here we might mention Dragon, Via-Voice, etc.*]

Another significant problem faced in a distributed voice-based system is a lack of uniformity/control in the speech recognition process. In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. These clients can take numerous forms (desktop PC, laptop PC, PDA, etc.) having varying speech signal processing and communications capability. Thus, from the server side perspective, it is not easy to assure uniform treatment of all users accessing a voice-enabled web page, since such users may have significantly disparate word recognition and error rate performances. Again, to enable such voice-based technologies on a wide-spread scale it is far more preferable to have a system that harmonizes and accounts for discrepancies in individual systems so that all users are able to interact in a satisfactory

manner with the remote server running the e-commerce, e-support and/or remote learning application.

[*Here we should talk about Qualcomm and GTE prior art on distributed processing*]

SUMMARY OF THE INVENTION

An object of the present invention, therefore, is to provide an improved system and method for overcoming the limitations of the prior art noted above;

A further object of the present invention is to provide ….;

A related object of the present invention is to provide ….

A further object of the present invention is to….

A system of the present invention therefore eliminates ….

The advantages of this approach are many, and include the fact that

One general aspect of the present invention, therefore, relates to a natural language query system (NLQS) that offers a fully interactive method for answering user's questions over a distributed network such as the Internet or a local intranet. This interactive system when implemented over the worldwide web (WWW) services of the Internet functions so that a client or user can ask a question in a natural language such as English, French, German or Spanish and receive the appropriate answer at his or her personal computer also in his or her native natural language.

The system is a distributed system consisting of a set of integrated software modules at the client's machine and another set of integrated software programs resident on a server or set of servers. The client-side software program is comprised of a speech recognition program, an agent and its control program, and a communication program. The server-side program is comprised of a communication program, a natural language engine (NLE), a database processor (DBProcess), an interface program for interfacing the DBProcess with the NLE, and a SQL database. In addition, the client's machine is equipped with a microphone and a speaker.

The system is specifically used to provide a single-best answer to a user's question. The question that is asked at the client's machine is articulated by the speaker and captured by a microphone that is built in as in the case of a notebook computer or is supplied as a standard attachment. Once the question is captured, the question is processed partially by NLQS client-side software resident in the client's machine. The output of this partial processing is a set of speech vectors that are transported to the server via the Internet to complete the recognition of the user's questions. This recognized speech is then converted to text at the server.

After the user's question is decoded by the speech recognition engine located at the server, the question is converted to a structured query language (SQL) query. This query is then simultaneously presented to a software process within the server called the DBProcess for preliminary processing and to the Natural Language Engine (NLE) module for extracting the noun phrases (NP) of the user's question. During the process of extracting the noun phrase within the NLE, the tokens of the users' question are tagged. The tagged tokens are then grouped so that the NP list can be determined. This information is stored and sent to the DBProcess process.

In the DBProcess, the SQL query is fully customized using the NP extracted from the user's question and other environment variables that are relevant to the application. For example, in a training application, the user's selection of course, chapter and or section would constitute the environment variables. The SQL query is constructed using the extended SQL Full-Text predicates - ***CONTAINS***, ***FREETEXT***, ***NEAR***, ***AND***. The SQL query is next sent to the Full-Text search engine within the SQL database, where a Full-Text search procedure is initiated. The result of this search procedure is recordset of answers. This recordset contains stored questions that are similar linguistically to the user's question. Each of these stored questions has a paired answer stored in a separate text file, whose path is stored in a table of the database.

The entire recordset of returned stored answers is then returned to the NLE engine in the form of an array. Each stored question of the array is then linguistically processed sequentially one by one. This linguistic processing constitutes the second step of a 2-step algorithm to determine the single best answer to the user's question. This second step proceeds as follows: for each stored question that is returned in the recordset, a NP of the stored question is compared with the NP of the user's question. After all stored questions of

the array are compared with the user's question, the stored question that yields the maximum match with the user's question is selected as the best possible stored question that matches the user's question. The metric that is used to determine the best possible stored question is the number of noun phrases.

The stored answer that is paired to the best-stored question is selected as the one that answers the user's question. The ID tag of the question is then passed to the DBProcess. This DBProcess the returns the answer which is stored in a file.

A communication link is again established to send the answer back to the client in compressed form. The answer once received by the client is decompressed and articulated to the user by the text-to-speech engine.

Computer-assisted instruction environments often require the assistance of mentors or live teachers to answer questions from students. This assistance often takes the form of organizing a separate pre-arranged forum or meeting time that is set aside for chat sessions or live call-in sessions so that at a scheduled time answers to questions may be provided. Because of the time immediacy and the on-demand or asynchronous nature of on-line training where a student may log on and take instruction at any time and at any location, it is important that answers to questions be provided in a timely and cost-effective manner so that the user or student can derive the maximum benefit from the material presented.

This invention addresses the above issues. It provides the user or student with answers to questions that are normally channeled to a live teacher or mentor. This invention provides a single-best answer to questions asked by the student. The student asks the question in his or her own voice in the language of choice. The speech is recognized and the answer to the question is found using a number of technologies including distributed speech recognition, full-text search database processing, natural language processing and text-to-speech technologies. The answer is presented to the user, as in the case of a live teacher, in an articulated manner by an agent that mimics the mentor or teacher, and in the language of choice - English, French, German, Japanese or other natural spoken language. The user can choose the agent's gender as well as several speech parameters such as pitch, volume and speed of the character's voice.

Other applications that benefit from NLQS are e-commerce applications. In this application, the user's query for a price of a book, compact disk or for the availability of any item that is to be purchased can be retrieved without the need to pick through various lists

on successive web pages. Instead, the answer is provided directly to the user without any additional user input.

Similarly, it is envisioned that this system can be used to provide answers to frequently-asked questions (FAQs). These questions are typical of a give web site and are provided to help the user find information related to a payment procedure or the specifications of a product. In all of these applications, the NLQS architecture can be applied.

….

Although the inventions are described below in a preferred embodiment involving ….., it will be apparent to those skilled in the art the present invention would be beneficially used in many environments where it is necessary to ….

<u>BRIEF DESCRIPTION OF THE DRAWINGS</u>

[*Ian: please give a simple explanation (1 sentence) of each drawing*]

DETAILED DESCRIPTION OF THE INVENTION

Overview of Inventions

As alluded to above, the present inventions allow a user to ask a question in a natural language such as English, French, German, Spanish or Japanese at a client computing system (which can be as simple as a personal digital assistant or cell-phone, or as sophisticated as a high end desktop PC) and receive an appropriate answer from a remote server also in his or her native natural language. As such, the embodiment of the invention shown in FIG. 1 is beneficially used in what can be generally described as a Natural Language Query System (NLQS) 100, which is configured to interact on a real-time basis to give a human-like dialog capability/experience for e-commerce, e-support, and e-learning applications.

The processing for NLQS 100 is generally distributed across a client side system 150, a data link 160, and a server-side system 180. These components are well-known in the art, and in a preferred embodiment include a personal computer system 150, an internet connection 160, and a larger scale computing system 180. It will be understood by those skilled in the art that these are merely exemplary components, and that the present invention is by no means limited to any particular implementation or combination of such systems. For example, client side system 150 could also be implemented as a computer peripheral, a PDA, as part of a cell-phone, as part of an internet adapted appliance, an internet linked kiosk, etc. Similarly, while an internet connection is depicted for data link 160, it is apparent that any channel that is suitable for carrying between data client system 160 and server 180 will suffice, including a LAN. Finally, it will be further appreciated that server system 180 may be a single, large scale system, or a collection of smaller systems interlinked to support a number of potential network users.

Initially speech input is provided in the form of a question or query articulated by the speaker at the client's machine or personal accessory. This speech input is captured and partially processed by NLQS client-side software 155 resident in the client's machine. To facilitate and enhance the human-like aspects of the interaction, the question is presented in the presence of an animated character 157 visible to the user who assists the user as a personal information retriever/agent. The agent can also interact with the user using both visible text output on a monitor/display (not shown) and/or in audible form using a text to speech engine 159. The output of the partial processing done by SRE 155 is a set of speech

vectors that are transmitted over communication channel 160 that links the user's machine or personal accessory to a server or servers via the Internet or a wireless gateway that is linked to the Internet as explained above. At server 180, the partially processed speech signal data is handled by a server side SRE 182, which then outputs recognized speech text corresponding to the user's question. Based on this user question related text, a text to query converter 184 formulates a suitable query that is used as input to a database processor 186. Based on the query, database processor 186 then locates and retrieves an appropriate answer using a customized SQL query from database 188. A Natural Language Engine 190 faciliates structuring the query to database 188. After a matching answer to the user's question is found, the former is transmitted in text form across data link 160, where it is converted into speech by text to speech engine 159, and thus expressed as oral feedback by animated character agent 157.

Because the speech processing is broken up in this fashion, it is possible to achieve real-time, interactive, human-like dialog consisting of a large, controllable set of questions/answers. The assistance of the animated agent 157 further enhances the experience, making it more natural and comfortable for even novice users. To make the speech recognition process more reliable, context-specific grammars and dictionaries are used, as well as natural language processing routines at NLE 190, to analyze user questions lexically. The text of the user's question is compared against text of other questions to identify the question posed by the user by DB processor/engine (DBE) 186. By optimizing the interaction and relationship of the SR engines 155 and 182, the NLP routines 190, and the dictionaries and grammers, an extremely fast and accurate match can be made, so that a unique and responsive answer can be provided to the user.

On the server side 180, interleaved processing further accelerates the speech recognition process. In simplified terms, the query is presented simultaneously both to NLE 190 after the query is formulated, as well as to DBE 186. NLE 190 and SRE 182 perform complementary functions in the overall recognition process. In general, SRE 182 is primarily responsible for determining the identity of the words articulated by the user, while NLE 190 is responsible for (*please fill in a one sentence statement here* ……………… ……………………………………………………………………..)

After the query is recognized by NLE 190 it is further processed again after preliminary search results are obtained from the question text database by the DBE. (*Please note that this is confusing: we need to clarify a little what happens with the feedback*)

Thus, the present invention uses a form of natural language processing (NLP) to achieve optimal performance in a speech based web application system. While NLP is known in the art, prior efforts in Natural Language Processing (NLP) work nonetheless have not been well integrated with Speech Recognition (SR) technologies to achieve reasonable results in a web-based application environment. In speech recognition, the result is typically a lattice of possible recognized words each with some probability of fit with the speech recognizer. In NLP, the input is always a sentence of words output by the SRE that has to be understood lexically, as opposed to being "recognized". [*Ian: we should elaborate on this difference*:] The NLP processes strings of words, rather than a lattice of words, since it is not possible in a practical manner to process a word lattice in a reasonable time. This is because of the complexity of the N words at any point being present in the lattice. A sequence of T words results in up to N^T possible distinct sentences.

As indicated earlier, although speech recognition technology has been available for several years, the technical requirements for the NLQS invention have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily. In realizing that even with the best of conditions, it might be not be possible to achieve the perfect 100% speech recognition accuracy that is required, the present invention employs an algorithm that balances the potential risk of the speech recognition process with the requirements of the natural language processing so that even in cases where perfect speech recognition accuracy is not achieved for each word in the query, the entire query itself is nonetheless recognized with sufficient accuracy.

This recognition accuracy is achieved even while meeting very stringent user constraints, such as short latency periods of 3 to 5 seconds (ideally – ignoring transmission latencies which can vary) for responding to a speech based query. This quick response time gives the overall appearance and experience of a real-time discourse that is more natural and pleasant from the user's perspective. Of course, non-real time applications can also benefit from the present teachings as well, since a centralized set of HMMs, grammars, dictionaries, etc., are maintained.

General Aspects of Speech Recognition Used in the Present Inventions

General background information on speech recognition can be found in the prior art references discussed above and incorporated by reference herein. Nonetheless, a discussion of some particular exemplary forms of speech recognition structures and techniques that are well-suited for NLQS 100 is provided next to better illustrate some of the characteristics, qualities and features of the present inventions.

Speech recognition technology is typically of two types – speaker independent and speaker dependent. In speaker-dependent speech recognition technology, each user has a voice file in which a sample of each potentially recognized word is stored. Speaker-dependent speech recognition systems typically have large vocabularies and dictionaries making them suitable for applications as dictation and text transcribing. It follows also that the memory and processor resource requirements for the speaker-dependent can be and are typically large and intensive.

Conversely speaker-independent speech recognition technology allows a large group of users to use a single vocabulary file. It follows then that the degree of accuracy that can be achieved is a function of the size and complexity of the grammars and dictionaries that can be supported for a given language. Given the context of applications for which NLQS, the use of small grammars and dictionaries allow speaker independent speech recognition technology to be implemented in NLQS.

The key issues or requirements for either type – speaker-independent or speaker-dependent, are accuracy and speed. As the size of the user dictionaries increase, the speech recognition accuracy metric – word error rate (WER) and the speed of recognition decreases. This is so because the search time increases and the pronunciation match becomes more complex as the size of the dictionary increases.

The basis of the NLQS speech recognition system is a series of Hidden Markov Models (HMM), which, as alluded to earlier, are mathematical models used to characterize any time varying signal. Because parts of speech are considered to be based on an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an outcome or observation can be

generated, according to an associated probability distribution. This finite state machine changes state once every time unit, and each time **t** such that a state **j** is entered, a spectral parameter vector $\mathbf{O}_t$ is generated with probability density $\mathbf{B}_j(\mathbf{O}_t)$**.** It is only the outcome, not the state which is visible to an external observer and therefore states are ``hidden" to the outside; hence the name Hidden Markov Model.

In HMM-based speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word can each be described by a Markov model. As with the general case, the Markov model when applied to speech also assumes a finite state machine which changes state once every time unit and each time that a state j is entered, a speech vector $\mathbf{o}_t$ is generated from the probability density $b_j(\mathbf{o}_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}

For a state sequence X, the joint probability that **O** is generated by the model M moving through a state sequence X is the product of the transition probabilities and the output probabilities. Only the observation sequence is known – the state sequence is hidden as mentioned before.

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences X = x(1), x(2), x(3) ,….. x(T), that is

$$P(\mathbf{O}|M) = \Sigma \{ a_{x(0)\, x(1)} \Pi\, b(x)\, (\mathbf{o}_t)\, a_{x(t)\, x(t+1)} \}$$

Given a set of models M_i, corresponding to words w_I equation 1 – 2 (*where is this?*) is solved by using 1-3 (*ditto?*) and also by assuming that:

$$P(\mathbf{O}|w_i) = P(\mathbf{O}|M_i)$$

All of this assumes that the parameters $\{a_{ij}\}$ and $\{b_j(\mathbf{o}_t)\}$ are known for each model M_i. This can be done, as explained earlier, by using a set of training examples corresponding to a particular model. Thereafter, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. So if a sufficient number of representative examples of each word are collected, then a HMM can be constructed which simply models all of the many sources of variability inherent in real speech. This training is well-known in the art, so it is not described at length herein, except to note that the distributed architecture of the present invention enhances the quality of HMMs, since they are derived and constituted at the server side, rather than the client side. In this way, appropriate samples from users of different geographical areas can be easily compiled and analyzed to optimize the possible variations expected to be seen across a particular language

to be recognized. Uniformity of the speech recognition process is also well-maintained, and error diagnotics are simplified, since each prospective user is using the same set of HMMs during the recognition process.

To determine the parameters of a HMM from a set of training samples, the first step typically is to make a rough guess as to what they might be. Then a refinement is done using the Baum-Welch estimation formulae. By these formulae, the maximum likelihood estimates of μ_j and Σ_j (*where μ_j is ……………….. and Σ_j is ………………………*) are:

$$\mu_j = \Sigma^{T}_{t=1} L_j (t)\, \mathbf{o}_t \,/\, [\Sigma^{T}_{t=1} L_j (t)\, \mathbf{o}_t]$$

A forward-backward algorithm is next used to calculate the probability of state occupation $L_j(t)$. If the forward probability $\alpha_j (t)$ for some model M with N states is defined as:

$$\alpha_j (t) = P(\mathbf{o}_{1,} \ldots\ldots, \mathbf{o}_{t,}\, x(t) = j \,|\, M)$$

This probability can be calculated using the recursion:

$$\alpha_j (t) = [\Sigma^{N-1}{}_{i=2}\, \alpha(t-1)\, a_{ij}] b_j (\mathbf{o}_t)$$

Similarly the backward probability can be computed using the recursion:

$$\beta_j (t) = \Sigma^{N-1}{}_{j=2}\, a_{ij}\, b_j(\mathbf{o}_{t+1})(t+1)$$

Realizing that the forward probability is a joint probability and the backward probability is a conditional probability, the probability of state occupation is the product of the two probabilities:

$$\alpha j (t)\, \beta_j (t) = P(\mathbf{O}, x(t) = j \,|\, M)$$

Hence the probability of being in state j at a time t is:

$$L_j(t) = 1/P\, [\alpha_j (t)\, \beta_j (t)]$$

$$\text{where } P = P(\mathbf{O} \,|\, M)$$

To generalize the above for continuous speech recognition, we assume the maximum likelihood state sequence where the summation is replaced by a maximum operation. Thus for a given model M, let ϕj (t) represent the maximum likelihood of observing speech vectors $\mathbf{o}_1$ **to** $\mathbf{o}_t$ and being used in state j at time t:

$$\phi_j (t) = \max\{\phi j\, (t)(t - 1)\alpha_{ij}\}\, \beta_j(\mathbf{o}_t)$$

Expressing this logarithmically to avoid underflow, this likelihood becomes:

$$\psi_j (t) = \max \{\psi_i (t - 1) + \log(\alpha_{ij})\} + \log(b_j(\mathbf{o}_t)$$

This is also known as the Viterbi algorithm. It can be visualized as finding the best path through a matrix where the vertical dimension represents the states of the HMM and horizontal dimension represents frames of speech i.e. time. To complete the extension to connected speech recognition, it is further assumed that each HMM representing the underlying sequence is connected. Thus the training data for continuous speech recognition should consist of connected utterances; however, the boundaries between words do not have to be known.

To improve computational speed/efficiency, the Viterbi algorithm is sometimes extended to achieve convergence by using what is known as a Token Passing Model. The token passing model represents a partial match between the observation sequence $\mathbf{o}_1$ **to** $\mathbf{o}_t$ and a particular model, subject to the constraint that the model is in state j at time t. This token passing model can be extended easily to connected speech environments as well if we allow the sequence of HMMs to be defined as a finite state network. A composite network that includes both phoneme-based HMMs and complete words can be constructed so that a single-best word can be recognized to form connected speech using word N-best extraction from the lattice of possibilities. This composite form of HMM-based connected speech recognizer is the basis of the NLQS speech recognizer module. Nonetheless, the present invention is not limited as such to such specific forms of speech recognizers, and can employ other techniques for speech recognition if they are otherwise compatible with the present architecture and meet necessary performance criteria for accuracy and speed to provide a real-time dialog experience for users.

The representation of speech for the present invention's HMM-based speech recognition system assumes that speech can be split into two components: a rapidly varying excitation signal (*representing* ………………..) and a slowly varying filtered signal (*representing* ………………..). In extracting the acoustic parameters from the user's speech input so that it can evaluated in light of a set of HMMs, cepstral analysis is used to separate the vocal tract information from the excitation information. The cepstrum of a signal is computed by taking the Fourier (or similar) transform of the log spectrum. The principal advantage of extracting cepstral coefficients is that they are de-correlated and the diagonal covariances can be used with HMMs. Since the human ear resolves frequencies non-linearly across the audio spectrum, it has been shown that a front-end that operates in a similar non-linear way improves speech recognition performance.

Accordingly, instead of a typical linear prediction-based analysis, the front-end of the NLQS speech recognition engine implements a simple, fast Fourier transform based filter bank designed to give approximately equal resolution on the Mel-scale. To implement this filter bank, a window of speech data is transformed using a software based Fourier transform and the magnitude taken. Each FFT magnitude is then multiplied by the corresponding filter gain and the results accumulated. The cepstral coefficients that are derived from this filter-bank analysis at the front end are calculated during a first partial processing phase of the speech signal by using a Discrete Cosine Transform of the log filter bank amplitudes. These cepstral coefficients are called Mel-Frequency Cepstral Coefficients (MFCC) and they represent some of the speech parameters transferred from the client side to characterize the acoustic features of the user's speech signal. These parameters are chosen for a number of reasons, including the fact that they can be quickly and consistently derived even across systems of disparate capabilities (i.e., for everything from a low power PDA to a high powered desktop system), they give good discrimination, they lend themselves to a number of useful recognition related manipulations, and they are relatively small and compact in size so that they can be transported rapidly across even a relatively narrow band link.

To augment the speech parameters an energy term in the form of the logarithm of the signal energy is added. Accordingly, RMS energy is added to the 12 MFCC's to make 13 coefficients. These coefficients together make up the partially processed speech data transmitted in compressed form from the user's client system to the remote server side.

The performance of the present speech recognition system is enhanced significantly by computing and adding time derivatives to the basic static MFCC parameters at the server side. These two other sets of coefficients -- the delta and acceleration coefficients representing change in each of the 13 values from frame to frame (actually measured across several frames), are computed during a second partial speech signal processing phase to complete the initial processing of the speech signal, and are added to the original set of coefficients after the latter are received. These MFCCs together with the delta and acceleration coefficients constitute the observation vector $\mathbf{O}_t$ mentioned above that is used for determining the appropriate HMM for the speech data.

The delta and acceleration coefficients are computed using the following regression formula:

$$d_t = \Sigma^{\theta}{}_{\theta=1} [c_{t+\theta} - c_{t-\theta}] / 2\Sigma^{\theta}{}_{\theta=1}\theta^2$$

where d_t is a delta coefficient at time t computed in terms of the corresponding static coefficients:

$$d_t = [c_{t+\theta} - c_{t-\theta}] / 2\theta$$

In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. In other words, both the first and second partial processing phases above are executed by the same DSP (or microprocessor) running a ROM or software code routine at the client's computing machine.

In constrast, because of several considerations, specifically - cost, technical performance, and client hardware uniformity, the present NLQS system uses a partitioned or distributed approach. While some processing occurs on the client side, the main speech recognition engine runs on a centrally located server or number of servers. More specifically, as noted earlier, capture of the speech signals, MFCC vector extraction and compression are implemented on the client's machine during a first partial processing phase. The primary MFCCs are then transmitted to the server over the channel, which, for example, can include a dial-up internet connection, a LAN connection, a wireless connection and the like.

After decompression, the delta and acceleration coefficients are computed at the server to complete the initial speech processing phase, and the resulting observation vectors $\mathbf{O}_t$ are also determined.

Speech Recognition Engine

The speech recognition engine is also located on the server, and is based on a HTK-based recognition network compiled from a word-level network, a dictionary and a set of HMMs. The recognition network consists of a set of nodes connected by arcs. Each node is either a HMM model instance or a word end. Each model node is itself a network consisting of states connected by arcs. Thus when fully compiled, a speech recognition network consists of HMM states connecnted by transitions. For an unknown inout utterance with T frames, every path from the start node to the exit node of the network passes through T HMM states. Each of these paths has log probability which is computed by summing the log probability of each individual transition in the path and th elog probability of each emitting

state generating the corresponding observation. The function of the Viterbi decoder is find those paths throuh the network which have the highest log probability. This is found uisng the Token Passing algorithm. In a network that has many nodes, the computation time is reduced by only allowing propagation of those tokens which will have some chance of becoming winners. This process is called pruning.

Natural Language Processor

In a typical natural language interface to a database, the user enters a question in his/her natural language for example English. The system parses is and translates it to a query language expression. The system then uses the query language expression to process the query and if the search is successful, a recordset representing the results is displayed in English either formatted as raw text or in a graphical form. The natural langauge interface for it to work well has a number of technical requirements. It needs to be robust – for example, in the sentence 'What's the departments turnover' it needs to decide that the word whats = what's = what is. And it has to determine that departments = department's. In addition to being robust, the natural language interface has to distinguish between the several possible forms of ambiguity that may exist in the natural language – lexical, structural, reference and ellipsis ambiguity. All of these requirements in addition to the general ability to perform the basic linguistic morphological operations of tokenization, tagging and grouping are implemented within the NLQS' Natural Language Engine. Tokenization is implemented by a text analyzer which treats the text as a series of tokens or useful meaningful units that are larger than individual characters, but smmaler than phrases and sentences. These include words, separable parts of words, and punctuation. Each token is associated with an offset and a length. The first phase of tokenization is the process of segmenttaion which extracts the individual tokens from the input text and keeps track of the offset where each token originated in the input text. The tokenizer ouput lists the offset and category for each token. In the next phase of the text analysis, the tagger uses a built-in morphological analyzer to look up each word/token in a phrase or sentence and interanlly lists all parts of speech. The output is the input string with each token tagged with a parts of speech notation. Finally the grouper which functions as aphrase extractor or phrase analyzer, determines which groups of words form phrases. These three operations which are the foundations for any modern linguistic processing schemes, are fully implmented in NLQS algorithms for determining the

single-best possible answer to the user's question.

SQL Database and Full-Text Query

Another key component of the NLQS system is the SQL-database. This database is used to store text, specifically the the answer-question pairs are stored in full-text tables of the database. Additionally, the full-text search capability of the database allows full-text searches to be carried out.

While a large portion of all digitally stored information is in the form of unstructured data, primarily text, it is now only possible to store this textual data in traditional relational database management systems (RDBMs) in character-based columns such as **varchar** and **text**. In order to effectively retrieve textual data from the database, techniques have to be implemented to issue queries against textual data and to retrieve the answers in a meaningful way where it provides the answers as in the case of the NLQS system. Traditional RDBMSs have not been designed for efficient full-text retrieval.

There are two major types of textual searches: Property - This search technology first applies filters to documents in order to extract properties such as author, subject, type, word count, printed page count, and time last written, and then issues searches against those properties; Full-text –this search technology first creates indexes of all non-noise words in the documents, and then uses these indexes to support linguistic searches and proximity searches.

The following two additional technologies are also implemented in this particular RDBMs:SQL Server also have been integrated: A Search service - a full-text indexing and search service that is called both index engine and search engine in the context of this document. Within the context of SQL Server, this is called full-text search; the parser component of the OLE DB Provider for Index Server 2.0 that accepts full-text SQL extensions and maps them into a form that can be processed by the search engine.

The four major aspects involved in implementing full-text retrieval of plain-text data from a full-text-capable RDBMs are: Managing the definition of the tables and columns that are registered for full-text searches; Indexing the data in registered columns - the indexing process scans the character streams, determines the word boundaries (this is called word breaking), removes all noise words (this also is called stop words), and then populates a full-text index with the remaining words; Issuing queries against registered columns for

populated full-text indexes; Ensuring that subsequent changes to the data in registered columns gets propagated to the index engine to keep the full-text indexes synchronized.

The underlying design principle for the indexing, querying, and synchronizing processes is the presence of a full-text unique key column (or single-column primary key) on all tables registered for full-text searches. The full-text index contains an entry for the non-noise words in each row together with the value of the key column for each row.

When processing a full-text search, the search engine returns to the RDBMs the key values of the rows that match the search criteria.

Unlike classic RDBMS indexes, traditional full-text indexes are not modified instantly when values in full-text registered columns are updated, when rows are added to full-text registered tables, or when rows are deleted from full-text registered tables. Rather, full-text indexes usually are repopulated asynchronously. There are two reasons for this:

- It typically takes significantly more time to update a full-text index than a classic index.
- Full-text searches usually are fuzzy by nature and so do not need to be as precise as classic searches.

During repopulation, the unique key column values are passed to the index engine to identify those items that need to be re-indexed. For example, if the title associated with V109 gets changed to “Mystery Island,” then the index should be modified to reflect this new value.

The full-text administration process starts by designating a table and its columns of interest for full-text search. Customized NLQS stored procedures are used first to register tables and columns as eligible for full-text search. After that, a separate request by means of a stored procedure is issued to populate the full-text indexes. [see other patent]

The result is that the underlying index engine gets invoked and asynchronous index population begins. Full-text indexing tracks which significant words are used and where they are located. For example, a full-text index might indicate that the word “NLQS” is found at word number 423 and word number 982 in the **Abstract** column of the **DevTools** table for the row associated with a **ProductID** of 6. This index structure supports an efficient search for all items containing indexed words as well as advanced search operations, such as phrase searches and proximity searches. (An example of a phrase search is looking for “white elephant,” where “white” is followed by “elephant.” An example of a proximity search is

looking for "big" and "house" where "big" occurs near "house.")

To prevent the full-text index from becoming bloated, noise words such as "a," "and," and "the" are ignored.

Extensions to the Transact-SQL language are used to construct full-text queries. The two key predicates that are used in the NLQS are **CONTAINS** and **FREETEXT.**

The **CONTAINS** predicate is used to determine whether or not values in full-text registered columns contain certain words and phrases. Specifically, this predicate is used to search for:

- A word or phrase.
- The prefix of a word or phrase.
- A word or phrase that is near another.
- A word that is an inflectional form of another (for example, "drive" is the inflectional stem of "drives," "drove," "driving," and "driven").
- A set of words or phrases, each of which is assigned a different weighting.

The **FREETEXT** predicate is a basic form of a natural language query. It is used to determine whether or not values in full-text registered columns reflect the meaning, rather than the exact words, specified in the predicate. Like a natural language processor, the index engine of RDBMS breaks the freetext string into a number of search terms, generates the stemmed form of the words, assigns heuristic weighting to each term, and then finds the matches. Any text, including words, phrases, or sentences, can be specified in the query. The RDBMS search engine matches values that reflect the meaning, rather than the exact wording, of the query.

The relational engine within SQL Server recognizes the **CONTAINS** and **FREETEXT** predicates and performs some minimal syntax and semantic checking, such as ensuring that the column referenced in the predicate has been registered for full-text searches. During query execution, a full-text predicate and other relevant information are passed to the full-text search component. After further syntax and semantic validation, the search engine is invoked and returns the set of unique key values identifying those rows in the table that satisfy the full-text search condition.

In addition to the **FREETEXT** and **CONTAINS**, other predicates such as **AND**, **LIKE**, **NEAR** are combined to create the customized NLQS SQL construct.

Full-Text Query Architecture of the SQL Database

The full-text query architecture is comprised of the following several components – Full-Text Query component, the SQL Server Relational Engine, the Full-Text provider and the Search Engine.

The **Full-Text Query** component of the SQL database accept a full-text predicate or rowset-valued function from the SQL Server; transform parts of the predicate into an internal format, and send it to RDBMS Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the resultset that is then returned to the submitter of the query.

The **SQL Server Relational Engine** accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the ft_search_condition and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query.

The **Full-Text Provider** parses and validates ft_search_condition, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy ft_search_condition. The handling of this rowset is conceptually similar to the code used in support of the **OPENROWSET()** and **OPENQUERY()** rowset-valued functions.

Although the present invention has been described in terms of a preferred embodiment, it will be apparent to those skilled in the art that many alterations and modifications may be made to such embodiments without departing from the teachings of the present invention. It will also be apparent to those skilled in the art that for purposes of the present discussion, the block diagram of the present invention has been simplified. The microcode and software routines executed by the host processor to effectuate the inventive methods may be embodied in various forms, including in a permanent magnetic media, a non-volatile ROM, a CD-ROM, or any other suitable machine-readable format. Accordingly, it is intended that the all such alterations and modifications be included within the scope and spirit of the invention as defined by the following claims.

What is claimed is:

DESCRIPTION OF SERVER SIDE SYSTEM 180

Introduction

A high level flow diagram of the set of preferred processes implemented on server side system 180 of Natural Language Query System 100 is illustrated in **Figs. 11A** through **Fig. 11C**. In a preferred embodiment, this process consists of a two step algorithm for completing the processing of the speech input signal, recognizing the meaning of the user's query, and retrieving an appropriate answer/response for such query.

The 1st step as illustrated in **Fig. 11A** can be considered a high-speed first-cut pruning mechanism, and includes the following operations: after completing processing of the speech input signal, the user's query is recognized at step **1101**, so that the text of the query is simultaneously sent to Natural Language Engine 190 (FIG. 1) at step **1107**, and to DB Engine 186 (also FIG.1) at step **1102**. By "recognized" in this context it is meant that the user's query is converted into a text string of distinct native language words through the HMM technique discussed earlier.

At NLE 190, the text string undergoes morphological linguistic processing at step **1108**: the string is tokenized the tags are tagged and the tagged tokens are grouped Next the noun phrases (NP) of the string are stored at **1109**, and also copied and transferred for use by DB Engine 186 during a DB Process at step **1110.** As illustrated in **Fig. 11A**, the string corresponding to the user's query which was sent to the DB Engine 186 at **1102**, is used together with the NP received from NLE 190 to construct an SQL Query at step **1103**. Next, the SQL query is executed at step **1104**, and a recordset of potential questions corresponding to the user's query are received as a result of a full-text search at **1105**, which are is then sent back to NLE 190 in the form of an array at step **1106**.

As can be seen from the above, this first step on the server side processing acts as an efficient and fast pruning mechanism so that the universe of potential "hits" corresponding to the user's actual query is narrowed down very quickly to a manageable set of likely candidates in a very short period of time.

Referring to **Fig. 11B**, in contrast to the first step above, the 2nd step can be considered as the more precise selection portion of the recognition process. It begins with linguistic processing of each of the stored questions in the array returned by the full-text

Deleted: ~~5.3~~ SERVER-SIDE
Deleted: The
Deleted: algorithm
Deleted: for the
Deleted: as implemented at the server
Deleted: 00
Deleted: 00
Deleted: T
Deleted: algorithm is a 2-step
Deleted: 00
Deleted: is as
Deleted: s
Deleted: the
Deleted: after being
Deleted: the NLE
Deleted: the DBProcess
Deleted: the
Deleted: corresponding to the query
Deleted: ,
Deleted: .
Deleted: is
Deleted: to the
Deleted: ¶
¶
Deleted: 00
Deleted: Process
Deleted: the
Deleted: the
Deleted: the
Deleted: from
Deleted: the
Deleted:
Deleted: the
Deleted: ¶
Deleted: 00
Deleted: of the 2-step algorithm
Deleted: the

Revised: 9/13/99

search process as possible candidates representing the user's query. Processing of these stored questions continues in NLE 190 as follows: each question in the array of questions corresponding to the recordset returned by the SQL full-text search undergoes morphological linguistic processing at step **1111**: in this operation, a text string corresponding to the retrieved candidate question is tokenized, the tags are tagged and the tagged tokens are grouped. Next noun phrases of the string are computed and stored at step **1112.** This process continues iteratively at point **1113, and the sequence of steps at 1118,1111, 1112, 1113** are repeated so that an NP for each retrieved candidate question is computed and stored. Once an NP is computed for all the retrieved candidate questions of the array, a comparison is made between each such retrieved candidate question and the user's query based on the magnitude of the NP value at step **1114**. This process is also iterative in that steps **1114**, **1115**, **1116**, **1119** are repeated so that the comparison of the NP for each retrieved candidate question with that of the NP of the user's query is completed. When there are no more stored questions in the array to be processed at step **1117**, the stored question that has the maximum NP relative to the user's query, is identified at **1117A** as the stored question which best matches the user's query.

Notably, it can be seen that the second step of the recognition process is much more computationally itensive than the first step above, because several text strings are tokenized, and a comparison is made of several NPs. This would not be practical, nonetheless, if it were not for the fact that the first step has already quickly and efficiently reduced the candidates to be evaluated to a significant degree. Thus, this more computationally intensive aspect of the present invention is extremely valuable, however because it yields extremely high accuracy in the overall query recognition process. In this regard, therefore, this second step of the query recognition helps to ensure the overall accuracy of the system, while the first step helps to maintain a satisfactory speed that provides a real-time feel for the user.

As illustrated in **Fig. 11C,** the last part of the query/response process occurs by providing an appropriate matching answer/response to the user. Thus, an identity of a matching stored question is completed at step 1120. Next a file path corresponding to an answer of the identified matching question is extracted at step **1121**. Processing continues so that the answer is extracted from the file path at **1122** and finally the answer is compressed and sent to client side system 150 at step **1123.**

Deleted: the
Deleted: also
Deleted:
Deleted: the
Deleted: stored
Deleted: the
Deleted: is
Deleted:
Deleted: the
Deleted: stored
Deleted: stored
Deleted: are computed
Deleted: stored
Deleted: stored
Deleted: ¶
Deleted: ¶
Deleted: 00
Deleted: the
Deleted: 1120
Deleted: the
Deleted: the
Deleted: the

Revised: 9/13/99

The discussion above is intended to convey a general overview of the primary components, operations, functions and characteristics of those portions of NLQS system 100 that reside on server side system 180. The discussion that follows describes in more detail the respective sub-systems.

Software modules used in Server Side System 180

The key software modules used on server-side system 180 of the NLQS system are illustrated in **Fig. 5**. These include generally the following components: a Communication module 500 – identified as CommunicationServer ISAPI **500A** (which is executed by SRE Server-side 182 – FIG. 1 and is explained in more detail below), and a database process DBProcess module **501** (executed by DB Engine 186 – FIG. 1). Natural language engine module 500C (executed by NLE 190 – FIG.1 and an interface 500B between the NLE process module 500C and the DBProcess module **500B**. As shown here, CommunicationServerISAPI **500A** includes a server-side speech recognition engine and appropriate communication interfaces required between client side system 150 and server side system 180. As further illustrated in **Fig. 5**, server-side logic of Natural Language Query System 100 also can be characterized as including two dynamic link library components: CommunicationServerISAPI **500** and DBProcess **501**. The CommunicationServerIASPI 500 is comprised of 3 sub-modules: Server-side Speech Recognition Engine module **500A**; Interface module 500B between Natural Language Engine modules 500C and DBProcess 501; and the Natural Language Engine modules **500C.**

DB Process **501** is a module whose primary function is to connect to a SQL database and to execute an SQL query that is composed in response to the user's query. In addition, this module interfaces with logic that fetches the correct answer from a file path once this answer is passed to it from the Natural Language Engine module **500C**.

[Note to Ian: it is my opinion that Fig.5 should be simplified to eliminate all the details within blocks 500a, 500b, 500c, etc., because they are duplicated, sometimes in inconsistent fashion, below.]

Speech Recognition Sub-System 182 on Server-Side System 180

The server side speech recognition engine module **500A** is a set of distributed components that perform the necessary functions and operations of speech recognition

Revised: 9/13/99

Deleted: each of the process steps of the NLQS algorithm in each respective sub-system
Deleted: as
Deleted: is comprised of several key processing modules
Formatted
Formatted
Deleted: the
Deleted: or
Formatted
Deleted: , the
Formatted
Deleted:) 9, 14,
Deleted: the
Deleted: The Communication module –
Deleted: the
Deleted: the
Deleted: the
Deleted: ¶
¶
Deleted: the
Deleted: the
Deleted: as
Deleted: consists of the following
Deleted: the
Deleted: ¶
the
Deleted: 1
Deleted: the
Deleted: the
Deleted: the
Deleted: . 500B;
Deleted: The
Deleted: ,
Deleted: the
Deleted: it
Formatted
Deleted: :
Deleted: of
Deleted: the

engine 182 (FIG.1) at server-side 180. These components can be implemented as software routines that are executed by server side 180 in conventional fashion. Referring to **Fig. 4A**, a more detailed break out of the operation of the speech recognition components at the server-side can be seen as follows: Within a portion **601** of the server side SRE module 500A, the binary MFCC vector byte stream corresponding to the speech signal's acoustic features extracted at client side system 150 and sent over the communication channel 160 is received. The MFCC acoustic vectors are decoded from the encoded HTTP byte stream as follows: Since the MFCC vectors contain embedded NULL characters, they cannot be transferred in this form to server side system 180 as such using HTTP protocol. Thus the MFCC vectors are first encoded at client-side 150 before transmission in such a way that all the speech data is converted into a stream of bytes without embedded NULL characters in the data. At the very end of the byte stream a single NULL character is introduced to indicate the termination of the speech utterance *(or for what other purpose?……………………..)*.

As explained earlier, to conserve latency time between the client and server, a smaller number of bytes (just the 13 MFCC coefficients) are sent from client side system 150 to server side system 180. This is done automatically for each platform to ensure uniformity, or can be tailored by the particular application environment – i.e., such as where it is determined that it will take less time to compute the delta and acceleration coefficients at the server (26 more calculations), than it would take to encode them at the client, transmit them, and then decode them from the HTTP stream. In general, since server side system 180 is usually better equipped to calculate the MFCC delta and acceleration parameters, this is a preferable choice. Furthermore, there is generally more control over server resources compared to the client's resources, which means that future upgrades, optimizations, etc., can be disseminated and shared by all to make overall system performance more reliable and predictable. So, the present invention can accommodate even the worst-case scenario where the client's machine may be quite thin and may just have enough resources to capture the speech input data and do minimal processing.

Dictionary Preparation & Grammar Files

Referring to **Fig. 4A**, within code block **605**, various options selected by the user (or gleaned from the user's status within a particular application) are received. For instance, in the case of a preferred remote learning system, Course, Chaptor and/or Section data items

Deleted: that reside at
Deleted: the
Deleted: -5-1
Deleted: signal flow for
Deleted: is
Deleted: ¶
¶
Formatted
Deleted: received
Deleted: the
Deleted: s
Deleted: the
Deleted: http
Deleted: the
Formatted
Deleted: -5-1
Deleted: the
Deleted: : Course, Chapter and/or section

Revised: 9/13/99

are communicated. In the case of other applications (such as e-commerce) other data options are communicated, such as the Product Class, Product Category, Product Brand, etc. loaded for viewing within his/her browser. These selected options are based on the context experienced by the user during an interactive process, and thus help to limit and define the scope – i.e. grammars and dictionaries that will be dynamically loaded to speech recognition engine 182 (FIG. 1) for Viterbi decoding during processing of the user speech utterance. For speech recognition to be optimized both grammar and dictionary files are used in a preferred embodiment. A Grammar file supplies the universe of available user queries; i.e., all the possible sentences that are to be recognized. The Dictionary file provides phonemes (the information of how to pronounce a word) of each word contained in the grammar file. It is apparent that if all the sentences for a given environment that can be recognized were contained in a single grammar file then recognition accuracy would be deteriorated and the loading time alone for such grammar and dictionary files would impair the speed of the speech recognition process.

To avoid these problems, specific grammars are loaded or actively configured as the current grammar according to the user's context, i.e., as in the case of a remote learning system, the Course, Chapter and/or Section selected. Thus the grammar and dictionary files are loaded dynamically according to the given Course, Chapter and/or Section as dictated by the user, or as determined automatically by an application program executed by the user. The second code block 602 implements the initialization of Speech Recognition engine 182 (FIG.1). The MFCC vectors received from client side system 150 along with the grammar filename and the dictionary file names are introduced to this block to initialize the speech decoder. As illustrated in **Fig. 4A**, the initialization process 602 uses the following sub-routines: A routine 602a for loading an SRE library. This then allows the creation of an object identified as External Source with code 602b using the received MFCC vectors. Code 602c allocates memory to hold the recognition objects. Routine 602d then also creates and initializes objects that are required for the recognition such as: Source, Coder, Recognizer and Results Loading of the Dictionary created by code **602e**, Hidden Markov Models (HMMs) generated with code **602f**; and Loading of the Grammar file generated by routine **602g**.

Speech Recognition **603** is the next routine invoked as illustrated in **Fig. 4A**, and is generally responsible for completing the processing of the user speech signals input on the

Deleted: the

Deleted: implemented, there has to be a

Deleted: The

Deleted: will be provided, while

Deleted: t

Deleted: I

Deleted: are

Deleted: the

Deleted: will

Deleted: speed of

Deleted: the

Deleted: made to be

Deleted: the

Deleted: of

Deleted: ¶
¶

Deleted: -5-1

Deleted: consists of

Deleted: processes

Deleted: ¶

Deleted: L

Deleted: the

Deleted: 602a

Deleted: External Source **602b**

Deleted: received

Deleted: It

Deleted: allocates the memory required

Deleted: 602c

Deleted: and

Deleted: the

Deleted: 602d

Deleted: and

Deleted: as illustrated in

Deleted: ¶

Deleted: step

Deleted:

Deleted: -5-1

Revised: 9/13/99

client side 150, which, as mentioned above, are preferably only partially processed (i.e., only MFCC vectors are computed during the first phase) when they are transmitted across link 160. Using the functions created in External Source by subroutine 602b, this code reads MFCC vectors, one at a time from an External Source **603a**, and processes them in block **603b** to realize the words in the speech pattern that are symbolized by the MFCC vectors captured at the client. During this second phase, an additional 13 delta coefficients and an additional 13 acceleration coefficients are computed as part of the recognition process to obtain the observation vectors $\mathbf{O}_t$ referred to earlier. Then, using a set of previously defined Hidden Markov Models (HMMs), the words corresponding to the user's speech utterance are determined in the manner described earlier. This completes the word "recognition" aspect of the query processing, which results are used further below to complete the query processing operations.

It will be appreciated by those skilled in the art that the distributed nature and rapid performance of the word recognition process, by itself, is extremely useful and may be implemented in connection with other environments that do not implicate or require additional query processing operations. For example, some applications may simply use individual recognized words for filling in data items on a computer generated form, and the aforementioned systems and processes can provide a rapid, reliable mechanism for doing so.

Once the user's speech is recognized, the flow of SRE 182 passes to Un-initialize SRE routine **604** where the speech engine is un-initialized as illustrated. In this block all the objects created in the initialization block are deleted by routine **604a**, and memory allocated in the initialization block during the initialization phase are removed by routine **604b**.

Again, it should be emphasized that the above are merely illustrative of embodiments for implementing the particular routines used on a server side speech recognition system of the present invention. Other variations of the same that achieve the desired functionality and objectives of the present invention will be apparent from the present teachings.

Database Processor 186 Operation – DBProcess

Construction of an SQL Query used as part of the user query processing is illustrated in **Fig. 4B**, A SELECT SQL statement is preferably constructed using a conventional **CONTAINS** predicate. Module **950** constructs the SQL query based on this SELECT SQL statement, which query is used for retrieving the best suitable question stored in the

Deleted:

Deleted: the

Deleted: block

Deleted: the

Deleted: ¶
¶

Deleted: the

Formatted

Formatted

Formatted

Formatted

Deleted: the

Deleted: as

Deleted: -5-4

Deleted: Construct

Deleted: Query

Deleted: the **SELECT**

Deleted: nd

Deleted: , is the module that

Revised: 9/13/99

database corresponding to the user's articulated query, (designated as Question here). A routine 951 then concatenates a table (what is is this???.........) name with the constructed **SELECT** statement. Next, the number of words present in each Noun Phrase of Question asked by the user is calculated by routine **952**. Then memory is allocated by routine **953** as needed to accommodate all the words present in the NP. Next a word List (identifying all the distinct words present in the NP) is obtained by routine 954. After this, this set of distinct words are concatenated by routine 955 to the SQL Query separated with a **NEAR ( )** keyword. Next, the **AND** keyword is concatenated to the SQL Query by routine 956 after each NP. Finally memory resources are freed by code 957 so as to allocate memory to store the words received from NP for any next iteration. Thus, at the end of this process, a completed SQL Query corresponding to the user's articulated question is generated. ***[Ian: I did not find any figure corresponding to this text, and, since it is redundant, I deleted it; if you think it still belongs, please let me know where the corresponding figure is]***

Connection to SQL Server – As illustrated in **Fig. 4C**, after the SQL Query is constructed by routine 710, a routine **711** implements a connection to the query database 717 to continue processing of the user query. The connection sequence and the subsequent retrieved recordset is implemented using the following routines:

1. Server and database names are assigned by routine 711A to a DBProcess member variable
2. A connection string is established by routine 711B;
3. The SQL Server database is connected under control of code **711C**
4. The SQL Query is received by routine **712A**
5. The SQL Query is executed by code **712B**
6. Extract the total number of records retrieved by the query
7. Allocate the memory to store the total number of paired questions
8. Store the entire number of paired questions into an array

1. ***[These boxes above should be identified in some simple way in the figures; also, items in box 716 need to be explained]***

[Ian: the remainder of FIG. 4C (used to be 4-5-2) should be given here]

Deleted: The
Deleted: is then concatenated
Deleted: 951
Deleted: first
Deleted: as much
Deleted: for
Deleted: the
Deleted: ¶
Deleted: 954
Deleted: Next,
Deleted: ese
Deleted: the
Deleted: 955
Deleted: 956
Deleted: the
Deleted: is
Deleted: the
Deleted: 957
Deleted: ¶ ¶ ... [1]
Formatted
Deleted: the
Deleted: -5-2
Deleted: the
Deleted: C
Deleted: D
Deleted: in the following steps
Deleted: the
Deleted: 711A
Formatted
Formatted: Bullets and Numbering
Formatted
Deleted: Connect t
Deleted: ,
Deleted: ,
Deleted: Execute the SQL query,
Formatted
Formatted
Formatted
Deleted: ¶ ... [2]
Deleted: Now referring back to F ... [3]
Formatted

Revised: 9/13/99

NLQS Database 188 - Table Organization

Figure 7 illustrates a preferred embodiment of a logical structure of tables used in a typical NLQS database 188 (FIG.1). When NLQS database 188 is used as part of NLQS query system 100 implemented as a remote learning/training environment, this database will include an organizational multi-level hierarchy that consists typically of a Course 7**01**, which is made of several chapters **702**, **703**, **704**. Each of these chapters can have one or more Sections **705**, **706**, **707** as shown for Chapter 1. A similar structure can exist for Chapter 2, Chapter 3 … Chapter N. Each section has a set of one or more question - answer pairs **708**, **709**, **710** stored in tables described in more detail below. While this is an appropriate and preferable arrangement for a training/learning application, it is apparent that other implementations would be possible and perhaps more suitable for other applications such as e-commerce, e-support, internet browsing, etc., depending on overall system parameters. It can be seen that the NLQS database 188 organization is intricately linked to the switched grammar architecture described earlier. In other words, the context (or environment) experienced by the user can be determined at any moment in time based at the selection made at the section level, so that only a limited subset of question-answer pairs 708 for example are appropriate for section 705. This in turn means that only a particular appropriate grammar for such question-answer pairs may be switched in for handling user queries while the user is experienceing such context. In a similar fashion, an e-commerce application for an internet based business may consist of a hierarchy that includes a first level "home" page 701 identifying user selectable options (product types, services, contact information, etc.), a second level may include one or more "product types" pages 702, 703, 704, a third page may include particular product models 705, 706, 707, etc., and with appropriate question-answer pairs 708, 709, and grammars customized for handling queries for such product models. Again, the particular implementation will vary from application to application, depending on the needs and desires of such business.

Table Organization

In a preferred embodiment, an independent database is used for each Course. Each database in turn can include three types of tables as follows: a Master Table as illustrated in

Deleted: The
Deleted: that
Deleted: the
Deleted: required in
Deleted: ¶ ¶
Deleted: to be
Deleted: ¶ ¶
Deleted: The
Deleted: shown
Deleted: in patent application.
Deleted: ¶
Deleted: There is
Deleted: specific
Deleted: s

Revised: 9/13/99

Figure 7A, at least one Chapter Table as illustrated in **Figure 7B** and at least one Section Table as illustrated in **Figure 7C**.

Deleted: s

As illustrated in **Fig. 7A,** a preferred embodiment of a Master Table has six columns – Field Name **701A**, Data Type **702A**, Size **703A**, Null **704A**, Primary Key **705A** and Indexed **706A**. These parameters are well-known in the art of database design and structure. The Master Table has only two fields – Chapter Name **707A** and Section Name **708A**. Both ChapterName and Section Name are commonly indexed.

Deleted: the

Deleted: CourseName

A preferred embodiment of a Chapter Table is illustrated in **Figure 7B**. As with the Master Table, the Chapter Table has six (6) columns – Field Name **720**, Data Type **721**, Size **722**, Null **723**, Primary Key **724** and Indexed **725**. There are nine (9) rows of data however, in this case, – Chapter_ID **726**, Answer_ID **727**, Section Name **728**, Answer_Title **729**, PairedQuestion **730**, AnswerPath **731**, Creator **732**, Date of Creation **733** and Date of Modification **734**.

Deleted: The

Deleted: T

An explanation of the Chapter Table fields is provided in **Figure 7C**. Each of the eight (8) Fields **730** has a description 731 and stores data corresponding to:

Deleted: This

Deleted: contains eight (8)

Deleted: as illustrated

AnswerID **732** – an integer that is automatically incremented for each answer given for user convenience

Section_Name **733** – the name of the section to which the particular record belongs This field along with the AnswerID is used as the primary key

Deleted: has to be made

Answer_Title **734** – A short description of the title of the answer to the user query

PairedQuestion **735** – Contains one or more combinations of questions for the related answers whose path is stored in the next column AnswerPath

AnswerPath **736** – contains the path of a file, which contains the answer to the related questions stored in the previous column; in the case of a pure question/answer application, this file is a text file, but, as mentioned above, could be a multi-media file of of any kind transportable over the data link 160

Deleted: the text

Creator **737**– Name of Content Creator

Date_of_Creation **738** – Date on which content was created

Date of Modification **739** – Date on which content was changed or modified

Revised: 9/13/99

A preferred embodiment of a Section Table is illustrated in **Figure 7D**. The Section Table has six (6) columns – Field Name **740**, Data Type **741**, Size **742**, Null **743**, Primary Key **744** and Indexed **745**. There are seven (7) rows of data – Answer_ID **746**, Answer_Title **747**, PairedQuestion **748**, AnswerPath **749**, Creator **750**, Date of Creation **751** and Date of Modification **752**. These names correspond to the same fields, columns already described above for the Master Table and Chapter Table.

Again, this is a preferred approach for the specific type of learning/training application described herein. Since the number of potential applications for the present invention is quite large, and each application can be customized, it is expected that other applications (including other learning/training applications) will require and/or be better accommodated by another table, column, and field structure/hierarchy.

[Ian: I think this discussion is duplicated below, so it is redundant here]

Full-Text Catalogs 1013

This is where full-text indexes reside. This set of catalogs 1013 is a file-system directory that is accessible only by an Administrator and Search Service 1010. Full-text indexes 1014 are organized into full-text catalogs, which are referenced by easy to handle names. Typically, full-text index data for an entire database is placed into a single full-text catalog.

[Note to Ian: I don't really think this next section is necessary, or even adds much to the gist of the inventive concepts]

Search Service and Search Engine – A query text search service is performed by an SQL Search System 1000 shown in FIG. 10. This system provides querying support to process full-text searches.

In general, SQL Search System determines which entries in a database index meet selection criteria specified by a particular text query that is constructed in accordance with an articulated user speech utterance. For each entry in the database that meets the selection criteria, a unique key column value and a ranking value are returned as well. *[Ian: this figure makes it look like there are two searches made, one in a relational database 1006, and one in a full-text*

Deleted: ¶

Deleted: the

Formatted

Deleted: **SQL Full-Text Search** ¶
The query components accept a full-text predicate or rowset-valued function from SQL Server, transform parts of the predicate into an internal format, and send it to the Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the result set that is then returned to the database processor¶
.¶
The SQL Server Relational Engine accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the *ft_search_condition* and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query.¶
¶
The Full-Text Provider parses and validates *ft_search_condition*, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is ... [4]

Formatted

Deleted: the

Deleted: The

Deleted: f

Deleted: friendly

Deleted: the

Formatted

Deleted: **Full-Text Indexing Administration**¶ ... [5]

Deleted: the

Deleted: S

Deleted: S

Deleted: the Search Engine. It performs two basic functions:¶ ... [6]

Deleted: es

Deleted: The Search Engine processes full-text search queries. It

Deleted: the

Deleted: the

Deleted: the value of the

Formatted

Revised: 9/13/99

index 1013; is this true, or is there something I do not understand here??? The use of DB 1006 is not explained below]

As illustrated in **Fig 1000**, a Full-Text Query Process is implemented as follows:

1. A query **1001** that uses a SQL full-text construct generated by DB procesor 186 is submitted to SQL Relational Engine 1002.
2. Queries containing either a **CONTAINS** or **FREETEXT** predicate are rewritten by routine **1003** so that a responsive rowset returned later from Full-Text Provider 1007 will be automatically joined to the table that the predicate is acting upon. This rewrite is amechanism used to ensure that these predicates are a seamless extension to a traditional SQL Server.
3. After this, Full-Text Provider 1007 is invoked, passing the following information for the query:
   a. A *ft_search_condition* parameter (this is a logical flag indicating a full text search condition)
   b. A friendly (what does this mean???...........) name of a full-text catalog where a full-text index of a table resides
   c. A locale ID to be used for language (for example, word breaking)
   d. Identities of a database, table, and column to be used in the query
   e. If the query is comprised of more than one full-text construct; when this is the caseFull-text provider 1007 is invoked separately for each construct.
4. SQL Relational Engine **1002** does not examine the contents of *ft_search_condition*. Instead, this information is passed along to Full-text provider **1007**, which verifies the validity of the query and then creates an appropriate internal representation of the full-text search condition.
5. The query reqeust/command is then passed to Querying Support **1011**.
6. Querying Support **1012** returns a rowset **1009** (from where????? Full-Text Catalog 1013?..................................) that contains unique key column values for any rows that match the full-text search criteria. A rank value also is returned for each row.

Deleted: the
Deleted: one of the
Deleted: s
Deleted: the
Deleted: the
Deleted: the
Deleted: the
Deleted: later
Deleted: the
Deleted: The
Deleted: The
Deleted: The
Deleted: the
Deleted: the
Deleted: The
Deleted: The
Deleted: i
Deleted: the
Deleted: , the
Deleted: f
Deleted: The
Deleted: the
Deleted: the
Deleted: f
Deleted: the
Deleted: the
Deleted: the

Revised: 9/13/99

7. The rowset of key column values 1009 is passed to SQL Relational Engine **1002.** If processing of the query implicates either a **CONTAINSTABLE()** or **FREETEXTTABLE()** function, **RANK** values are returned; otherwise, any rank value is filtered out.

8. The rowset values 1009 are plugged into the initial query with values obtained from relational database 1006, and a result set **1015** is then returned for further processing to yield a response to the user.

At this stage of the query recognition process, the speech utterance by the user has been rapidly converted into an carefully crafted text query, and this text query has been initially processed so that an initial matching set of results can be further evaluated for a final determination of the appropriate matching question/answer pair.

[Ian: I think the text below is redundant, given the discussion further down the road. Let me know if you feel otherwise]

Interface between NLE 190 and DB Processor 188

The result set 1015 of candidate questions corresponding to the user query utterance are presented to NLE 190 for further processing as shown in FIG. 4D to determine a "best" matching question/answer pair. An NLE/DBProcessor interface module coordinates the handling of user queries, analysis of noun-phrases (NPs) of retrieved questions sets from the SQL query based on the user query, comparing the retrieved question NPs with the user query NP, etc. between NLE 190 and DB Processor 188. So, this part of the server side code contains functions, which interface processes resident in both NLE block 190 and DB Processor block 188. The functions are illustrated in **Fig. 4D**; As seen here, code routine **800** implements functions to extract the Noun Phrase (NP) list from the user's question. This part of the code interacts with NLE 190 and gets the list of Noun Phrases in a sentence articulated by the user. Similarly, Routine **813** retrieves an NP list from the list of corresponding candidate/paired questions 1015 and stores these questions into an (ranked

Deleted: 1005

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: ¶

Formatted

Deleted: The the recordset containing paired questions is passed again to the NLE. This step is the 2nd step of the 2-step algorithm as illustrated in **Fig. 8B**. Its carries out detailed linguistic analysis of the paired questions and in addition iteratively compares the NP of each record of the recordset with that of the NP of the user's question to identify the stored question that best matches the user's question. The criterion against which the best stored question is selected is the maximum number of NP, that is, the paired question that has the maximal number of NP is chosen to be the one that best matches the user's question. ¶

¶

As illustrated in **Fig. 5**, after the recordset is returned from the database, and the array of paired questions are stored as an array, this array is then passed to **14** of the NLE. The sequence of events that follow are similar to that used to extract the NP of the user's question. In this analysis, the noun phrases are extracted from each of the paired questions in sequence.¶

¶

As illustrated in **Fig 5**, the entry point for the array of paired questions to the NLE is port **14** of the NLE. The processing steps that follow are via several sub-blocks labeled **9a** through **9e**. Each of these sub-blocks implement the several different linguistic functions performed by the NLE that are required to extract the noun phrases from the array of paired questions. The steps are as follows:¶

¶ ... [7]

Formatted

Deleted: Process

Deleted: T

Deleted: the

Deleted: s

Deleted: -5-3.

Deleted: that

Deleted: s

Deleted: is a function, which

Deleted: the

Deleted: This function is used to find out the Noun Phrases existed in the ... [8]

Deleted: the

Revised: 9/13/99

by NP value?) array. Thus, at this point, NP data has been generated for the user query, as well as for the candidate questions 1015.

Next, a function identified as Get Best Answer ID 815 is implemented. This part of the code gets a best answerID corresponding to the user's query. To do this, routines 813A, 813B first find out the number of Noun phrases for each entry in the retrieved set 1015 that match with the Noun phrases in the user's query. Then routine 815a selects a final result record from the candidate retrieved set 1015 that contains the maximum number of matching Noun phrases.

The ID corresponding to the best answer corresponding to the selected final result record question is then generated by routine 815 which then returns it to DB Process shown in FIG.4C. As seen there, a Best Answer ID I is received by routine 716A, and used by a routine 716B to retrieve an answer file path. Routine 716C then opens and reads the answer file, and communicates the substance of the same to routine 716D. The latter then compresses the answer file data, and sends it over data link 160 to client side system 150 for processing as noted earlier (i.e., to be rendered into audible feedback, visual text/graphics, etc.). Again, in the context of a learning/instructional application, the answer file may consist solely of a single text phrase, but in other applications the substance and format will be tailored to a specific question in an appropriate fashion. For instance, an "answer" may consist of a list of multiple entries corresponding to a list of responsive category items (i.e., a list of books to a particular author) etc. Other variations will be apparent depending on the particular environment.

Natural Language Engine 190

Again referring to **Fig. 4D**, the general structure of NL engine 190 is depicted. This engine implements the word analysis or morphological analysis of words that make up the user's query, as well as phrase analysis of phrases extracted from the query.

As illustrated in **Fig. 8A**, the functions used in a morphological analysis include tokenizers **802A**, stemmers **804A** and morphological analyzers **806A**. The functions that

Deleted: This part of the code is a function, which interacts with the NLE and gets the list of Noun phrases in a sentence. This function is used to find out the Noun phrases existed in the paired questions that are similar to the user question

Deleted: the

Deleted: -

Deleted: as illustrated **815**.

Deleted: the

Deleted: This function

Deleted: s

Deleted: are matching

Deleted:

Deleted: it

Deleted: the

Deleted: , which

Deleted: ¶
Then a comparison of the number of NP in the User's question with that of NP list of paired questions is implemented **815a** . This part of the code is a function which is used by block **815a** of the code to compare the find out the number of matching Noun Phrases in the given user's query and a given paired question which is similar to the user asked question.

Deleted: returned

Deleted: to the Get Best Answer ID **815**,

Deleted: the

Deleted: block

Deleted: -5-3

Deleted: is

Deleted: and

Deleted: the

Deleted: that make up the

Revised: 9/13/99

comprise the phrase analysis include tokenizers, taggers and groupers, and their relationship is shown in FIG.8.

Tokenizer 802A is a software module that functions to break up text of an input sentence 801A into a list of tokens **803A**. In performing this function, tokenizer 802A goes through input text 801A and treats it as a series of tokens or useful meaningful units that are typically larger than individual characters, but smaller than phrases and sentences. These tokens 803A can include words, separable parts of word and punctuation. Each token 803A is given an offset and a length. The first phase of tokenization is segmentation, which extracts the individual tokens from the input text and keeps track of the offset where each token originated from in the input text. Next categories are associated with each token, based on its shape (what is this???.......). The process of tokenization is well-known in the art, so it can be performed by any convenient application suitable for the present invention.

Following tokenization, a stemmer process 804A is executed, which can includetwo separate forms – inflectional and derivational, for analyzing the tokens to determine their respective stems 805A. Aninflectional stemmer recognizes affixes and returns the word which is the stem. A derivational stemmer on the other hand recognizes derivational affixes and returns the root word or words. ***[Ian: 806a and 806b are not explained here, but should be]***

As illustrated in **Fig. 8**, phrase analysis 800 is the next step that is performed after tokenization. Tokens 803 are assigned to parts of a speech tag by a tagger routine **804**, and a grouper routine **806** recognizes groups of words as phrases of a certain syntactic type. These syntactic types include for example the noun phrases mentioned earlier, but could include other types if desired. Specifically, tagger 804 is a parts-of-speech disambiguator, which analyzes words in context. It has a built-in morphological analyzer (not shown) that allows it to identify all possible parts of speech for each token. The output of tagger 804 is a string with each token tagged with a parts-of-speech label 805. The final step in the linguistic process 800 is the grouping of words to form phrases 807. This function is performed by the grouper **806**, and is very dependent, of course, on the performance and output of tagger component 804.

Deleted: The
Deleted: t
Deleted: n
Deleted: that
Deleted: s
Deleted: the
Deleted: the
Deleted: associated with
Deleted:
Deleted: do
Deleted: the
Deleted: might be of
Deleted: es
Deleted: for its
Deleted: **805A**
Deleted: The
Deleted: The
Formatted
Deleted: The t
Deleted: a
Deleted: the
Deleted: the
Deleted: the
Deleted:
Deleted:
Deleted: 's
Deleted: the
Deleted:
Deleted:
Deleted: n
Deleted: the

Revised: 9/13/99

The processes of tokenization and phrase analysis are well-known in the art, so they can be performed by any convenient application suitable for the present invention.

Accordingly, at the end of linguistic processing 800, a list of noun phrases (NP) 807 is generated in accordance with the user's query utterance . This set of NPs generated by NLE 190 helps significantly to refine the search for the best answer, so that a single-best answer can be later provided for the user's question.

[FIG.8B should really be discussed here to explain what it shows]

The particular components of NLE 190 are shown in FIG. 4D, and include several components. Each of these components implement the several different functions required in NLE 190 as now explained.

Initialize Grouper Resources Object and the Library **900** – this routine initializes the structure variables required to create grouper resource object and library. Specifically, it initializes a particular natural language used by NLE 190 to create a Noun Phrase, for example the English natural language is initialized for a system that serves the English language market. In turn, it also creates the objects (routines?) required for Tokenizer, Tagger and Grouper (discussed above) with routines 900A, 900B, 900C and 900D respectively, and initializes these objects with appropriate values. It also allocates memory to store all the recognized Noun Phrases for the retrieved question pairs.

Tokenizing of the words from the given text (from the query or the paired questions) is performed with routine **909B** – here all the words are tokenized with the help of a dictionary. The resultant tokenized words are passed to a Tagger routine 909C.

At routine 909C, tagging of all the tokens is done **and** and the output is passed to a Grouper routine 909D.

The Grouping of all tagged token to form NP list is implemented by routine 909D so that the Grouper groups all the tagged token words and outputs the Noun Phrases.

Deleted: ¶
The general linguistic functions implemented by the NLE are illustrated in **Fig 8**. The input text **801** is first tokenized **802** to produce a list of tokens **803.** Then the list of tokens is then tagged by the Tagger **804** to produce the different parts of speech **805** for each token. Next each parts of speech is grouped by the grouper **806** to produce the list of phrases --

Deleted:

Deleted: for example,

Deleted: The key purpose of the

Deleted: module is to

Deleted: is

Deleted: using linguistic processing

Formatted

Deleted: entry point of

Deleted: 0

Deleted: is made up of

Deleted: **9a** through **9e**

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: the

Deleted: object

Deleted: The T

Deleted: **909C**. Here the Tagger

Deleted: object tags all the tokenized words

Deleted: object

Deleted: 909D

Deleted: here

Deleted: object

Revised: 9/13/99

Un-initializing of the grouper resources object and freeing of the resources, is peformed by routines 909EA, 909EB and 909EC. i These include Token Resources, Tagger Resources and Grouper Resources respectively. After initialization, the resources are freed. The memory that was used to store all Noun Phrases are also de-allocated.

Deleted: 9E

Deleted: n this component all the initialized resources are un-initialized.

Deleted: 909EA

Deleted: 909EA,

Deleted: Grouper

Deleted: 909EC.

Deleted: and the

Deleted: ¶
¶
¶
¶
¶

Revised: 9/13/99

Page 7: [1] Deleted J. Nicholas Gross 11/2/1999 8:36:00 PM

As illustrated in **Fig. 5**, using all the words present in the Noun Phrase along with Database name (Course), Table name (Chapter and/or Section) chosen by the user at client side, the SQL Query is constructed **10**. It also uses full-text predicate **CONTAINS** and other predicates such as **NEAR( )** and the **AND** operator while preparing the SQL Query. This customized SQL Query is then passed to the DBProcess **12** so that the paired questions can be retrieved from the NLQS database.

Page 7: [2] Deleted J. Nicholas Gross 11/2/1999 9:00:00 PM

In connecting to the SQL Server itself and to the database that is specified in the code of the DBProcess, the following 3 steps are executed:

The connection object is prepared

The connection string is prepared using the database name, user name, password, and the server name, **711A**

Then a connection is made to the SQL Server database, **711C**

Page 7: [3] Deleted J. Nicholas Gross 11/2/1999 9:00:00 PM

Now referring back to **Fig. 5**, the next step is the Execution of the SQL Query **12B**. To execute the SQL Query, the SQL Query constructed in Block **10** is passed to **712a** of **Fig. 4-5-2**. After assembling there, it is passed to block **712b** where is executed.

Then there is the creation of code in this same block **712b** that creates a recordset object so that it accommodate the recordset after it is returned from the NLQS database.

Page 10: [4] Deleted J. Nicholas Gross 11/3/1999 2:41:00 PM

SQL Full-Text Search

The query components accept a full-text predicate or rowset-valued function from SQL Server, transform parts of the predicate into an internal format, and send it to the Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the result set that is then returned to the database processor

.

The SQL Server Relational Engine accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the *ft_search_condition* and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query.

The Full-Text Provider parses and validates *ft_search_condition*, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy *ft_search_condition*. The handling of this rowset is conceptually similar to the code used in support of the **OPENROWSET()** and **OPENQUERY()** rowset-valued functions.

Page 10: [5] Deleted J. Nicholas Gross 11/3/1999 1:19:00 PM

Full-Text Indexing Administration

The following steps are started by an administrator using GUIs or stored procedures (either on demand or as scheduled). First, enable the database for full-text search and then identify the tables and columns that are to be registered for full-text search. This information is stored in the SQL Server system tables. When a table is activated for full-text processing, a population start seed for that table is sent to indexing support. Next, request the initial population of the full-text index for a table. Actually, the granularity of population is a full-text catalog, so if more than one table has been linked to a full-text catalog, the result is the full population of the full-text indexes for all tables linked to that catalog.

The knowledge of the tables and rows that require indexing resides in SQL Server, so when indexing support receives a population request for a full-text catalog, it calls back into SQL Server to obtain data from all the columns in the table that have been marked for indexing. When this data arrives, it is passed to the index engine where it is broken into words. Noise words are removed and the remaining words are stored in the index. Full-text indexes are kept current by using a GUI that sets up a schedule for periodic refreshes of a full-text catalog. This GUI uses stored procedures from the SQL Server job scheduler. It also is

possible to request a refresh at any time, either by means of a GUI or by direct use of a stored procedure.

If a table has a row-versioning (timestamp) column, repopulation can be handled more efficiently. At the time the population start seed for a table is constructed, the largest row-versioning value in the database is remembered. When an incremental population is requested, the SQL Server handler connects to the database and requests only rows where the row-versioning value is greater than the remembered value.
During an incremental population, if there is a full-text indexed table in a catalog that does not have a row-versioning column, then that table will be completely repopulated.

There are two cases where a complete re-population is performed, even though there is a row-versioning column on the table. These are:

In tables when the schema has changed.

In tables activated since the last population.

After populating tables with a row-versioning column, the remembered row-versioning value is updated. Since incremental repopulation on relatively static tables with timestamp columns can be completed faster than a complete repopulation, users can schedule repopulation on a more frequent basis. For a given database, users should take care not to mix index data from tables with timestamp columns with tables in the same full-text catalog, because these two groups of tables usually should be on separate repopulation schedules.

Additional tables and columns will be registered for full-text search, and full-text indexes will be generated for them. The full-text search capability may be removed from some tables and columns. Some full-text catalogs may be dropped. This step may be repeated several times.

Page 10: [6] Deleted J. Nicholas Gross 11/3/1999 1:31:00 PM

the Search Engine. It performs two basic functions:

Indexing support accepts requests to populate the full-text index of a given table.

Querying

Page 12: [7] Deleted J. Nicholas Gross 11/3/1999 4:25:00 PM

The the recordset containing paired questions is passed again to the NLE. This step is the 2[nd] step of the 2-step algorithm as illustrated in **Fig. 8B**. Its carries out detailed linguistic analysis of the paired questions and in addition iteratively compares the NP of each record of the recordset with that of the NP of the user's question to identify the stored question that best matches the user's question. The criterion against which the best stored question is selected is the maximum number of NP, that is, the paired question that has the maximal number of NP is chosen to be the one that best matches the user's question.

As illustrated in **Fig. 5**, after the recordset is returned from the database, and the array of paired questions are stored as an array, this array is then passed to **14** of the NLE. The sequence of events that follow are similar to that used to extract the NP of the user's question. In this analysis, the noun phrases are extracted from each of the paired questions in sequence.

As illustrated in **Fig 5**, the entry point for the array of paired questions to the NLE is port **14** of the NLE. The processing steps that follow are via several sub-blocks labeled **9a** through **9e**. Each of these sub-blocks implement the several different linguistic functions performed by the NLE that are required to extract the noun phrases from the array of paired questions. The steps are as follows:

Initialize Grouper Resources Object and the Library **9a**, this block initializes the structure variables required to create the grouper resource object and library. Specifically, it initializes the natural language used by the NLE to create the Noun Phrase, for example the English natural language is initialized for the system that serves the English language market. In turn, it creates the objects required for Tokenizer, Tagger and Grouper and initializes these objects with appropriate values. It also allocates the memory to store all the recognized Noun Phrases.

Tokenizing of the words from the given text **9b** - in this block all the words are tokenized with the help of dictionary. The resultant tokenized words are passed to the Tagger object.

The Tagging of all the tokens are implemented in **9c**. Here the Tagger object will tag all the tokenized words and the output is passed to Grouper object.

The Grouping of all tagged token to form NP list **9d** - in this block the Grouper object groups all the tagged token words and gives out as Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources 9e – in this block all the initialized resources are un-initialized and the resources are freed. Also de-allocates the memory allocated to store all Noun Phrases

Again referring to **Fig. 5**, at the end of this processing, the NP list is returned from the NLE to block **13** of **Fig. 5**. Specifically, it is passed to the block **15** – Get Best Answer ID. Before it extracts the identification (ID) for the single best answer, processing passes to Block **15a** where a comparison of the NP of user's question with NP of the Paired Questions extracted from database is made to find out the best suitable question present in the database that matches the user's question.

Block **15** then passes the Best Answer ID to block **16** of **Fig. 5.** Some preparatory steps are executed before processing takes place in **block 16**. These steps are as follows:

Creates a database process DLL class object

Gets the NP list from the question and builds the full-text SQL query using the NP list from the question and section name

Gets all the paired questions related to the user's questions from the database. If no records are fetched, then an error is returned. If only one record is returned, then the answer is returned using the answer's file path from the fetched single record. Otherwise, the NLE is called to get the single best record.

*As illustrated in **Fig. 4-5-2**, block **716a** of this diagram receives the "best record number". The file path for this "best record number" is then fetched in block **716b.** Next that particular file is opened in block **716c**. Also in the same block **716c**, the contents of the file are read and the contents passed to block **716d**.*

In determining the file path, the following are the steps:

The record pointer is moved to the specified record number.

The path of the answer file is next retrieved

The content from the file is read

The answer is returned

Once the answer is returned, the file is sent to block ***716d****, where it is compressed. The actual sequence of steps involved are as follows:*

The size of the answer to be compressed is obtained

Sufficient memory is allocated to hold the compressed answer.

The compressed answer and the size of the original answer is returned to the client.

Page 12: [8] Deleted J. Nicholas Gross 11/3/1999 4:39:00 PM

This function is used to find out the Noun Phrases existed in the user query

Subj: **Revised Draft**
Date: 11/8/99 6:50:06 AM Pacific Standard Time
From: imb@best.com (Ian Bennett)
To: NicdaGreek@aol.com

File: First draft corrected on 11_07_99.doc (120320 bytes)
DL Time (46666 bps): < 1 minute

Hi Nick:

Attached please find the edits and revisions that I made to the main draft.

I went over the document carefully and followed your suggestions - adding stuff where you needed explanations and deleted stuff that was unnecessary.

I also took a pass at writing the abstracts. Please edit where necessary, for example one or two of them may be too wordy.

All of my corrections are in blue with deletions noted as strikeouts

The client-side detailed description looks good. I need to go through the server-side description next and will send both along with the revised drawings ASAP.

Ian

Hi Nick:

Attached please find the edits and revisions that I made to the maindraft.

I went over the document carefully and followed your suggestions - addingstuff where you needed explanations and deleted stuff that wasunnecessary.

I also took a pass at writing the abstracts. Please edit where necessary,for example one or two of them may be too wordy.

All of my corrections are in blue with deletions noted as strikeouts

The client-side detailed description looks good. I need to go through theserver-side description next and will send both along with the reviseddrawings ASAP.

Ian

----------------------- Headers -------------------------------

Return-Path:
Received: from rly-zc03.mx.aol.com (rly-zc03.mail.aol.com [172.31.33.3]) by air-zc05.mail.aol.com (v62.15) with ESMTP; Mon, 08 Nov 1999 09:50:06 -0500
Received: from proxy2.ba.best.com (proxy2.ba.best.com [206.184.139.14]) by rly-zc03.mx.aol.com (v62.10) with ESMTP; Mon, 08 Nov 1999 09:49:35 -0500
Received: from imb (imb.vip.best.com [204.156.155.72])
by proxy2.ba.best.com (8.9.3/8.9.2/best.out) with ESMTP id GAA17812
for ; Mon, 8 Nov 1999 06:47:54 -0800 (PST)
Message-Id: <4.2.0.58.19991108064521.00ab5a30@shell11.ba.best.com>
X-Sender: imb@shell11.ba.best.com
X-Mailer: QUALCOMM Windows Eudora Pro Version 4.2.0.58
Date: Mon, 08 Nov 1999 06:53:20 -0800
To: NicdaGreek@aol.com
From: Ian Bennett
Subject: Revised Draft
Mime-Version: 1.0
Content-Type: multipart/mixed;
boundary="=====================_131211808==_"

Express Mail Label No. US......................

DISTRIBUTED SPEECH-BASED INTERACTIVE SYSTEM
AND METHOD THEREFOR

FIELD OF THE INVENTION

The invention relates to a system and an interactive method for responding to speech based user inputs and queries presented over a distributed network such as the Internet or local intranet. This interactive system when implemented over the world-wide web services (WWW) of the Internet, functions so that a client or user can ask a question in a natural language such as English, French, German, Spanish or Japanese and receive the appropriate answer at his or her computer or accessory also in his or her native natural language. The system has particular applicability to such applications as remote learning, e-commerce, technical e-support services, Internet searching, etc.

BACKGROUND OF THE INVENTION

The Internet, and in particular, the World-Wide Web (WWW), is growing in popularity and usage for both commercial and recreational purposes, and this trend is expected to continue. This phenomenon is being driven, in part, by the increasing and widespread use of personal computer systems and the availability of low cost Internet access. The emergence of inexpensive Internet access devices and high speed access techniques such as ADSL, cable modems, satellite modems, and the like, are expected to further accelerate the mass usage of the WWW.

Accordingly, it is expected that the number of entities offering services, products, etc., over the WWW will increase dramatically over the coming years. Until now, however, the internet "experience" for users has been limited mostly to non-voice based input/output devices, such as keyboards, intelligent electronic pads, mice, trackballs, printers, monitors, etc. This presents somewhat of a bottleneck for interacting over the WWW for a variety of reasons.

First, there is the issue of familiarity. Many kinds of applications lend themselves much more naturally and fluently to a voice-based environment. For instance, most people shopping for audio recordings are very comfortable with asking a live sales clerk in a record store for information on titles by a particular author, where they can be found in the store, etc. While it is often possible to browse and search on one's own to locate items of interest, it is usually easier and more efficient to get some form of human assistance first, and, with few exceptions, this request for assistance is presented in the form of a oral query. In addition, many persons cannot or will not, because of

physical or psychological barriers, use any of the aforementioned conventional I/O devices. For example, many older persons cannot easily read the text presented on WWW pages, or understand the layout/hierarchy of menus, or manipulate a mouse to make finely coordinated movements to indicate their selections. Many others are intimidated by the look and complexity of computer systems, WWW pages, etc., and therefore do not attempt to use online services for this reason as well.

Thus, applications which can mimic normal human interactions are likely to be preferred by potential on-line shoppers and persons looking for information over the WWW. It is also expected that the use of voice-based systems will increase the universe of persons willing to engage in e-commerce, e-learning, etc. To date, however, there are very few systems, if any, which permit this type of interaction, and, if they do, it is very limited. For example, various commercial programs sold by IBM (Via Voice) and Kurzweil (Dragon) permit some user control of the interface (opening, closing files) and searching (by using previously trained URLs) but they do not present a flexible solution that can be used by a number of users across multiple cultures and without time consuming voice training.

Another issue presented by the lack of voice-based systems is efficiency. Many companies are now offering technical support over the Internet, and some even offer live operator assistance for such queries. While this is very advantageous (for the reasons mentioned above) it is also extremely costly and inefficient, because a real person must be employed to handle such queries. This presents a practical limit that results in long wait times for responses or high labor overheads. In general, a service presented over the WWW is far more desirable if it is "scalable," or, in other words, able to handle an increasing amount of user traffic with little if any perceived delay or troubles by a prospective user.

In a similar context, while remote learning has become an increasingly popular option for many students, it is practically impossible for an instructor to be able to field questions from more than one person at a time. Even then, such interaction usually takes place for only a limited period of time because of other instructor time constraints. To date, however, there is no practical way for students to continue a human-like question and answer type dialog after the learning session is over, or without the presence of the instructor to personally address such queries.

Conversely, another aspect of emulating a human-like dialog involves the use of oral feedback. In other words, many persons prefer to receive answers and information in audible form. While a form of this functionality is used by some websites to communicate information to visitors,

it is not performed in a real-time, interactive question-answer dialog fashion so its effectiveness and usefulness is limited.

Yet another area that could benefit from speech-based interaction involves so-called "search" engines used by Internet users to locate information of interest at web sites, such as the those available at *Yahoo.com*, *metacrawler.com*, *Excite.com*, etc. These tools permit the user to form a search query using either combinations of keywords or metacategories to search through a web page database containing text indices associated with one or more distinct web pages. After processing the user's request, therefore, the search engine returns a number of hits which correspond, generally, to URL pointers and text excerpts from the web pages that represent the closest match made by such search engine for the particular user query based on the search processing logic used by search engine. The structure and operation of such prior art search engines, including the mechanism by which they build the web page database, and parse the search query, are well known in the art. To date, applicant is unaware of any such search engine that can easily and reliably search and retrieve information based on speech input from a user.

There are a number of reasons why the above environments (e-commerce, e-support, remote learning, internet searching, etc.) do not utilize speech-based interfaces, despite the many benefits that would otherwise flow from such capability.

First, there is obviously a requirement that the output of the speech recognizer be as accurate as possible. One of the more reliable approaches to speech recognition used at this time is based on the Hidden Markov Model (HMM) – a model used to mathematically describe any time series. Because speech is considered to have an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of *states*, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called *transition probabilities*. In a particular state an outcome or *observation* can be generated, according to the associated probability distribution. This finite state machine changes state once every time unit, and each time t such that a state j is entered, a spectral parameter vector $\mathbf{O}_t$ is generated with probability density $B_j(\mathbf{O}_t)$. It is only the outcome, not the state visible to an external observer and therefore states are "hidden" to the outside; hence the name Hidden Markov Model. [*Here we should make a reference to some contemporary patents, articles that describe HMM … **DONE***]. The basic theory of HMMs was published in a series of classic papers by Baum and his colleagues in the late 1960's and early 1970's. HMMs were first used in speech applications by Baker at Carnegie

Mellon, by Jelenik and colleagues at IBM in the late 1970's and by Steve Young and colleagues at Cambridge University, UK in the 1990's. Some typical papers and texts are as follows:

1. L.E. Baum, T. Petrie, "Statistical inference for probabilistic functions for finite state Markov chains", Ann. Math. Stat., 37: 1554-1563, 1966
2. L.E. Baum, "An inequality and associated maximation technique in statistical estimation for probabilistic functions of Markov processes", Inequalities 3: 1-8, 1972
3. J.H. Baker, "The dragon system – An Overview", IEEE Trans. on ASSP Proc., ASSP-23(1): 24-29, Feby. 1975
4. F. Jeninek et al, "Continuous Speech Recognition: Statistical methods" in Handbook of Statistics, II, P.R. Kristnaiad, Ed. Amsterdam, The Netherlands, North-Holland, 1982
5. L.R. Bahl, F. Jeninek, R.L. Mercer, "A maximum likelihood approach to continuous speech recognition", IEEE Trans. Pattern Anal. Mach. Intell., PAMI-5: 179-190, 1983
6. J.D. Ferguson, "Hidden Markov Analysis: An Introduction", in Hidden Markov Models for Speech, Institute of Defense Analyses, Princeton, NJ. 1980.
7. H.R. Rabiner and B.H. Juang, "Fundamentals of Speech Recognition", Prentice Hall, 1993
8. H.R. Rabiner, "Digital Processing of Speech Signals", Prentice Hall, 1978

More recently research has progressed in extending HMM and combining HMMs with neural networks to speech recognition applications at various laboratories. The following is a representative paper:

9. Nelson Morgan, Hervé Bourlard, Steve Renals, Michael Cohen and Horacio Franco (1993), Hybrid Neural Network/Hidden Markov Model Systems for Continuous Speech Recognition. *Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7, No. 4 pp. 899-916. Also in I. Guyon and P. Wang editors, *Advances in Pattern Recognition Systems using Neural Networks,* Vol. 7 of a Series in Machine Perception and Artificial Intelligence. World Scientific, Feb. 1994.

While the HMM-based speech recognition ~~approach~~ yields very good results, contemporary variations of this technique cannot guarantee a word accuracy requirement of 100% exactly and consistently, as will be required for WWW applications for all possible all user and environment

conditions. Thus, although speech recognition technology has been available for several years, and has improved significantly, the technical requirements have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily. ***[Note: we should explain generally what natural language process is, and some examples of its use in the art …*** *DONE*]. Natural language processing (NLP) is concerned with the parsing, understanding and indexing of transcribed utterances and larger linguistic units. Because spontaneous speech contains many surface phenomena such as disfluencies, - hesitations, repairs and restarts, discourse markers such as '*well*' and other elements which cannot be handled by the typical speech recognizer, it is the problem and the source of the large gap that separates speech recognition and natural language processing technologies. Except for silence between utterances, another problem is the absence of any marked punctuation available for segmenting the speech input into meaningful units such as utterances. For optimal NLP performance, these types of phenomena should be annotated at its input. However, most continuous speech recognition systems, produce only a raw sequence of words.

Second, most of the very reliable voice recognition systems are speaker-dependent, requiring that the interface be "trained" with the user's voice, which takes a lot of time, and is thus very undesirable from the perspective of a WWW environment, where a user may interact only a few times with a particular website. Furthermore, speaker-dependent systems usually require a large user dictionary (one for each unique user) which reduces the speed of recognition. This makes it much harder to implement a real-time dialog interface with satisfactory response capability (i.e., something that mirrors normal conversation – on the order of 3 – 5 seconds is probably ideal). [*Here we might mention Dragon, Via-Voice, etc DONE*.] At present, the typical shrink-wrapped speech recognition application software include offering s from IBM (ViaVoice) and Dragon Systems (Dragon). While most of these applications are adequate for dictation and other transcribing applications, they are woefully inadequate for applications such as NLQS where the word error rate must be close to 0%. In addition these offerings require long training times and are typically are non client-server configurations.

Another significant problem faced in a distributed voice-based system is a lack of uniformity/control in the speech recognition process. In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. These clients can take numerous forms (desktop PC, laptop PC, PDA, etc.) having varying speech signal processing and

communications capability. Thus, from the server side perspective, it is not easy to assure uniform treatment of all users accessing a voice-enabled web page, since such users may have significantly disparate word recognition and error rate performances. Again, to enable such voice-based technologies on a wide-spread scale it is far more preferable to have a system that harmonizes and accounts for discrepancies in individual systems so that all users are able to interact in a satisfactory manner with the remote server running the e-commerce, e-support and/or remote learning application.

[*Here we should talk about Qualcomm and GTE prior art on distributed processing* - *DONE*]

The two significant patents that cover prior art in this area are US Patents 5956683 and 5960399. US Patent 5956683 – Distributed Voice Recognition System issued to Qualcomm describes an implementation of distributed voice recognition between a telephony-based handset and a remote station. In this implementation, the acoustic feature apparatus resides at the remote station. However, the patent describes the benefits that result from locating of the system for acoustic feature extraction at the portable or cellular phone in order to limit degradation of the acoustic features due to quantization distortion resulting from the narrow bandwidth telephony channel.

Another patent - US Patent 5960399 – Client/Server Speech Processor/Recognizer issued to GTE describes the implementation of a HMM-based distributed speech recognition system. It departs significantly in that it does not state any techniques for optimizing the acoustic feature extraction apparatus. Most importantly, the server-based recognizer only recognizes the user's speech and simply returns certain keywords such as user's name and travel destination to fill out a dedicated form on the user's machine. Also the streaming of the acoustic parameters are not implemented in real-time and streaming takes place after silence is detected.

<u>SUMMARY OF THE INVENTION</u>

An object of the present invention, therefore, is to provide an improved system and method for overcoming the limitations of the prior art noted above;

A further object of the present invention is to provide ….;

A related object of the present invention is to provide ….

A further object of the present invention is to….

A system of the present invention therefore eliminates ….

The advantages of this approach are many, and include the fact that

One general aspect of the present invention, therefore, relates to a natural language query system (NLQS) that offers a fully interactive method for answering user's questions over a distributed network such as the Internet or a local intranet. This interactive system when implemented over the worldwide web (WWW) services of the Internet functions so that a client or user can ask a question in a natural language such as English, French, German or Spanish and receive the appropriate answer at his or her personal computer also in his or her native natural language.

The system is a distributed system consisting of a set of integrated software modules at the client's machine and another set of integrated software programs resident on a server or set of servers. The client-side software program is comprised of a speech recognition program, an agent and its control program, and a communication program. The server-side program is comprised of a communication program, a natural language engine (NLE), a database processor (DBProcess), an interface program for interfacing the DBProcess with the NLE, and a SQL database. In addition, the client's machine is equipped with a microphone and a speaker.

The system is specifically used to provide a single-best answer to a user's question. The question that is asked at the client's machine is articulated by the speaker and captured by a microphone that is built in as in the case of a notebook computer or is supplied as a standard attachment. Once the question is captured, the question is processed partially by NLQS client-side software resident in the client's machine. The output of this partial processing is a set of speech vectors that are transported to the server via the Internet to complete the recognition of the user's questions. This recognized speech is then converted to text at the server.

After the user's question is decoded by the speech recognition engine located at the server, the question is converted to a structured query language (SQL) query. This query is then simultaneously presented to a software process within the server called the DBProcess for preliminary processing and to the Natural Language Engine (NLE) module for extracting the noun phrases (NP) of the user's question. During the process of extracting the noun phrase within the NLE, the tokens of the users' question are tagged. The tagged tokens are then grouped so that the NP list can be determined. This information is stored and sent to the DBProcess process.

In the DBProcess, the SQL query is fully customized using the NP extracted from the user's question and other environment variables that are relevant to the application. For example, in a

training application, the user's selection of course, chapter and or section would constitute the environment variables. The SQL query is constructed using the extended SQL Full-Text predicates - ***CONTAINS***, ***FREETEXT***, ***NEAR***, ***AND***. The SQL query is next sent to the Full-Text search engine within the SQL database, where a Full-Text search procedure is initiated. The result of this search procedure is recordset of answers. This recordset contains stored questions that are similar linguistically to the user's question. Each of these stored questions has a paired answer stored in a separate text file, whose path is stored in a table of the database.

The entire recordset of returned stored answers is then returned to the NLE engine in the form of an array. Each stored question of the array is then linguistically processed sequentially one by one. This linguistic processing constitutes the second step of a 2-step algorithm to determine the single best answer to the user's question. This second step proceeds as follows: for each stored question that is returned in the recordset, a NP of the stored question is compared with the NP of the user's question. After all stored questions of the array are compared with the user's question, the stored question that yields the maximum match with the user's question is selected as the best possible stored question that matches the user's question. The metric that is used to determine the best possible stored question is the number of noun phrases.

The stored answer that is paired to the best-stored question is selected as the one that answers the user's question. The ID tag of the question is then passed to the DBProcess. This DBProcess the returns the answer which is stored in a file.

A communication link is again established to send the answer back to the client in compressed form. The answer once received by the client is decompressed and articulated to the user by the text-to-speech engine.

Computer-assisted instruction environments often require the assistance of mentors or live teachers to answer questions from students. This assistance often takes the form of organizing a separate pre-arranged forum or meeting time that is set aside for chat sessions or live call-in sessions so that at a scheduled time answers to questions may be provided. Because of the time immediacy and the on-demand or asynchronous nature of on-line training where a student may log on and take instruction at any time and at any location, it is important that answers to questions be provided in a timely and cost-effective manner so that the user or student can derive the maximum benefit from the material presented.

This invention addresses the above issues. It provides the user or student with answers to questions that are normally channeled to a live teacher or mentor. This invention provides a single-

best answer to questions asked by the student. The student asks the question in his or her own voice in the language of choice. The speech is recognized and the answer to the question is found using a number of technologies including distributed speech recognition, full-text search database processing, natural language processing and text-to-speech technologies. The answer is presented to the user, as in the case of a live teacher, in an articulated manner by an agent that mimics the mentor or teacher, and in the language of choice - English, French, German, Japanese or other natural spoken language. The user can choose the agent's gender as well as several speech parameters such as pitch, volume and speed of the character's voice.

Other applications that benefit from NLQS are e-commerce applications. In this application, the user's query for a price of a book, compact disk or for the availability of any item that is to be purchased can be retrieved without the need to pick through various lists on successive web pages. Instead, the answer is provided directly to the user without any additional user input.

Similarly, it is envisioned that this system can be used to provide answers to frequently-asked questions (FAQs). These questions are typical of a give web site and are provided to help the user find information related to a payment procedure or the specifications of a product. In all of these applications, the NLQS architecture can be applied.

….

Although the inventions are described below in a preferred embodiment involving ….., it will be apparent to those skilled in the art the present invention would be beneficially used in many environments where it is necessary to ….

BRIEF DESCRIPTION OF THE DRAWINGS

[*Ian: please give a simple explanation (1 sentence) of each drawing* … DONE]

The following are the list of drawings:

FIG. 1: Overall NLQS system diagram

FIG. 2: Client-side – System Logic Diagram

FIG. 2-2: Client-side Initialization

FIG.2-2A: Client-side Initialization of Speech Recognition Engine

FIG.2-2B: Client-side Initialization of MS Agent

The following are the list of tables:

DETAILED DESCRIPTION OF THE INVENTION

Overview of Inventions

As alluded to above, the present inventions allow a user to ask a question in a natural language such as English, French, German, Spanish or Japanese at a client computing system (which can be as simple as a personal digital assistant or cell-phone, or as sophisticated as a high end desktop PC) and receive an appropriate answer from a remote server also in his or her native natural language. As such, the embodiment of the invention shown in FIG. 1 is beneficially used in what can be generally described as a Natural Language Query System (NLQS) 100, which is configured to interact on a real-time basis to give a human-like dialog capability/experience for e-commerce, e-support, and e-learning applications.

The processing for NLQS 100 is generally distributed across a client side system 150, a data link 160, and a server-side system 180. These components are well known in the art, and in a preferred embodiment include a personal computer system 150, an Internet connection 160A, 160B, and a larger scale computing system 180. It will be understood by those skilled in the art that these are merely exemplary components, and that the present invention is by no means limited to any particular implementation or combination of such systems. For example, client-side system 150 could also be implemented as a computer peripheral, a PDA, as part of a cell-phone, as part of an Internet-adapted appliance, an Internet linked kiosk, etc. Similarly, while an Internet connection is depicted for data link 160A, it is apparent that any channel that is suitable for carrying data between client system 150 and server system 180 will suffice, including a LAN. Finally, it will be further appreciated that server system 180 may be a single, large-scale system, or a collection of smaller systems interlinked to support a number of potential network users.

Initially speech input is provided in the form of a question or query articulated by the speaker at the client's machine or personal accessory. This speech input is captured and partially processed by NLQS client-side software 155 resident in the client's machine. To facilitate and enhance the human-like aspects of the interaction, the question is presented in the presence of an animated character 157 visible to the user who assists the user as a personal information retriever/agent. The agent can also interact with the user using both visible text output on a monitor/display (not shown) and/or in audible form using a text to speech engine 159. The output of the partial processing done by SRE 155 is a set of speech vectors that are transmitted over communication channel 160 that links the user's machine or personal accessory to a server or

servers via the Internet or a wireless gateway that is linked to the Internet as explained above. At server 180, the partially processed speech signal data is handled by a server-side SRE 182, which then outputs recognized speech text corresponding to the user's question. Based on this user question related text, a text-to-query converter 184 formulates a suitable query that is used as input to a database processor 186. Based on the query, database processor 186 then locates and retrieves an appropriate answer using a customized SQL query from database 188. A Natural Language Engine 190 facilitates structuring the query to database 188. After a matching answer to the user's question is found, the former is transmitted in text form across data link 160B, where it is converted into speech by text to speech engine 159, and thus expressed as oral feedback by animated character agent 157.

Because the speech processing is broken up in this fashion, it is possible to achieve real-time, interactive, human-like dialog consisting of a large, controllable set of questions/answers. The assistance of the animated agent 157 further enhances the experience, making it more natural and comfortable for even novice users. To make the speech recognition process more reliable, context-specific grammars and dictionaries are used, as well as natural language processing routines at NLE 190, to analyze user questions lexically. The text of the user's question is compared against text of other questions to identify the question posed by the user by DB processor/engine (DBE) 186. By optimizing the interaction and relationship of the SR engines 155 and 182, the NLP routines 190, and the dictionaries and grammars, an extremely fast and accurate match can be made, so that a unique and responsive answer can be provided to the user.

On the server side 180, interleaved processing further accelerates the speech recognition process. In simplified terms, the query is presented simultaneously both to NLE 190 after the query is formulated, as well as to DBE 186. NLE 190 and SRE 182 perform complementary functions in the overall recognition process. In general, SRE 182 is primarily responsible for determining the identity of the words articulated by the user, while NLE 190 is responsible for the linguistic morphological analysis of both the user's query and the search results returned after the database query.

After the user's query is analyzed by NLE 190 some parameters are extracted and sent to the DBProcess. Additional statistics are stored in an array for the 2nd step of processing. During the 2nd step of 2-step algorithm, the recordset of preliminary search results are sent to the NLE 160 for processing. At the end of this 2nd step, the single question that matches the user's query is sent to the DBProcess where further processing yields the paired answer that is paired with the single best

stored question. (*Please note that this is confusing: we need to clarify a little what happens with the feedback --- DONE*)

Thus, the present invention uses a form of natural language processing (NLP) to achieve optimal performance in a speech based web application system. While NLP is known in the art, prior efforts in Natural Language Processing (NLP) work nonetheless have not been well integrated with Speech Recognition (SR) technologies to achieve reasonable results in a web-based application environment. In speech recognition, the result is typically a lattice of possible recognized words each with some probability of fit with the speech recognizer. As described before, the input to a typical NLP system is typically a large linguistic unit. The NLP system is then charged with the parsing, understanding and indexing of this large linguistic unit or set of transcribed utterances. The result of this NLP process is to understand lexically or morphologically the entire linguistic unit as opposed to word recognition. Put another way, the linguistic unit or sentence of connected words output by the SRE ~~that~~ has to be understood lexically, as opposed to being "recognized". [*Ian: we should elaborate on this difference … DONE*] ~~The NLP processes strings of words, rather than a lattice of words, since it is not possible in a practical manner to process a word lattice in a reasonable time. This is because of the complexity of the N words at any point being present in the lattice. A sequence of T words results in up to N^T possible distinct sentences.~~

As indicated earlier, although speech recognition technology has been available for several years, the technical requirements for the NLQS invention have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily. In realizing that even with the best of conditions, it might be not be possible to achieve the perfect 100% speech recognition accuracy that is required, the present invention employs an algorithm that balances the potential risk of the speech recognition process with the requirements of the natural language processing so that even in cases where perfect speech recognition accuracy is not achieved for each word in the query, the entire query itself is nonetheless recognized with sufficient accuracy.

This recognition accuracy is achieved even while meeting very stringent user constraints, such as short latency periods of 3 to 5 seconds (ideally – ignoring transmission latencies which can vary) for responding to a speech-based query. This quick response time gives the overall appearance and experience of a real-time discourse that is more natural and pleasant from the user's perspective. Of course, non-real time applications can also benefit from the present teachings as well, since a centralized set of HMMs, grammars, dictionaries, etc., are maintained.

General Aspects of Speech Recognition Used in the Present Inventions

General background information on speech recognition can be found in the prior art references discussed above and incorporated by reference herein. Nonetheless, a discussion of some particular exemplary forms of speech recognition structures and techniques that are well-suited for NLQS 100 is provided next to better illustrate some of the characteristics, qualities and features of the present inventions.

Speech recognition technology is typically of two types – speaker independent and speaker dependent. In speaker-dependent speech recognition technology, each user has a voice file in which a sample of each potentially recognized word is stored. Speaker-dependent speech recognition systems typically have large vocabularies and dictionaries making them suitable for applications as dictation and text transcribing. It follows also that the memory and processor resource requirements for the speaker-dependent can be and are typically large and intensive.

Conversely speaker-independent speech recognition technology allows a large group of users to use a single vocabulary file. It follows then that the degree of accuracy that can be achieved is a function of the size and complexity of the grammars and dictionaries that can be supported for a given language. Given the context of applications for which NLQS, the use of small grammars and dictionaries allow speaker independent speech recognition technology to be implemented in NLQS.

The key issues or requirements for either type – speaker-independent or speaker-dependent, are accuracy and speed. As the size of the user dictionaries increase, the speech recognition accuracy metric – word error rate (WER) and the speed of recognition decreases. This is so because the search time increases and the pronunciation match becomes more complex as the size of the dictionary increases.

The basis of the NLQS speech recognition system is a series of Hidden Markov Models (HMM), which, as alluded to earlier, are mathematical models used to characterize any time varying signal. Because parts of speech are considered to be based on an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an outcome or observation can be generated, according to an associated probability distribution. This finite state machine changes state once every time unit, and each time **t** such that a state **j** is entered, a spectral

parameter vector $\mathbf{O}_t$ is generated with probability density $\mathbf{B}_j(\mathbf{O}_t)$**.** It is only the outcome, not the state which is visible to an external observer and therefore states are ``hidden" to the outside; hence the name Hidden Markov Model.

In isolated speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word can each be described by a Markov model as follows:

$$\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, \ldots\ldots\ldots \mathbf{o}_T \qquad (1\text{-}1)$$

where $\mathbf{o}_t$ is a speech vector observed at time t. The isolated word recognition then is to compute:

$$\arg\max \{ P(w_i | \mathbf{O})\} \qquad (1\text{-}2)$$

By using Bayes' Rule,

$$\{ P(w_i | \mathbf{O})\} = [P(\mathbf{O} | w_i)\, P(w_i)] \,/\, P(\mathbf{O}) \qquad (1\text{-}3)$$

In the general case, the Markov model when applied to speech also assumes a finite state machine which changes state once every time unit and each time that a state j is entered, a speech vector $\mathbf{o}_t$ is generated from the probability density $b_j(\mathbf{o}_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}

For a state sequence X, the joint probability that **O** is generated by the model M moving through a state sequence X is the product of the transition probabilities and the output probabilities. Only the observation sequence is known – the state sequence is hidden as mentioned before.

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences X = x(1), x(2), x(3) ,….. x(T), that is

$$P(\mathbf{O}|M) = \Sigma \{ a_{x(0)\,x(1)} \Pi\, b(x)\, (\mathbf{o}_t)\, a_{x(t)\,x(t+1)} \}$$

Given a set of models M_i, corresponding to words w_i equation 1-2 (*where is this? …* DONE) is solved by using 1-3 (*ditto? …* DONE) and also by assuming that:

$$P(\mathbf{O}|w_i) = P(\mathbf{O}|M_i)$$

All of this assumes that the parameters $\{a_{ij}\}$ and $\{b_j(\mathbf{o}_t)\}$ are known for each model M_i. This can be done, as explained earlier, by using a set of training examples corresponding to a particular model. Thereafter, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. So if a sufficient number of representative examples of each word are collected, then a HMM can be constructed which simply models all of the many sources of variability inherent in real speech. This training is well-known in the art, so it is not described at

length herein, except to note that the distributed architecture of the present invention enhances the quality of HMMs, since they are derived and constituted at the server side, rather than the client side. In this way, appropriate samples from users of different geographical areas can be easily compiled and analyzed to optimize the possible variations expected to be seen across a particular language to be recognized. Uniformity of the speech recognition process is also well-maintained, and error diagnostics are simplified, since each prospective user is using the same set of HMMs during the recognition process.

To determine the parameters of a HMM from a set of training samples, the first step typically is to make a rough guess as to what they might be. Then a refinement is done using the Baum-Welch estimation formulae. By these formulae, the maximum likelihood estimates of μ_j ~~and~~ $\cancel{\Sigma_j}$ (where μ_j is mean vector and Σ_j is covariance matrix) ~~are~~ is:

$$\mu_j = \Sigma^{T}_{t=1} L_j (t) \mathbf{o_t} / [\Sigma^{T}_{t=1} L_j (t) \mathbf{o_t}]$$

A forward-backward algorithm is next used to calculate the probability of state occupation $L_j(t)$. If the forward probability $\alpha_j (t)$ for some model M with N states is defined as:

$$\alpha_j (t) = P(\mathbf{o_1}, \ldots\ldots, \mathbf{o_t}, x(t) = j | M)$$

This probability can be calculated using the recursion:

$$\alpha_j (t) = [\Sigma^{N-1}_{i=2} \alpha(t-1) a_{ij}] b_j (\mathbf{o_t})$$

Similarly the backward probability can be computed using the recursion:

$$\beta_j (t) = \Sigma^{N-1}_{j=2} a_{ij} b_j(\mathbf{o_{t+1}})(t+1)$$

Realizing that the forward probability is a joint probability and the backward probability is a conditional probability, the probability of state occupation is the product of the two probabilities:

$$\alpha j (t) \beta_j (t) = P(\mathbf{O}, x(t) = j | M)$$

Hence the probability of being in state j at a time t is:

$$L_j(t) = 1/P [\alpha_j (t) \beta_j (t)]$$

$$\text{where } P = P(\mathbf{O} | M)$$

To generalize the above for continuous speech recognition, we assume the maximum likelihood state sequence where the summation is replaced by a maximum operation. Thus for a given model M, let $\phi j (t)$ represent the maximum likelihood of observing speech vectors $\mathbf{o_1}$ **to** $\mathbf{o_t}$ and being used in state j at time t:

$$\phi_j (t) = \max\{\phi j (t)(t - 1)\alpha_{ij}\} \beta_j(\mathbf{o_t})$$

Expressing this logarithmically to avoid underflow, this likelihood becomes:

$$\psi_j(t) = \max\{\psi_i(t-1) + \log(\alpha_{ij})\} + \log(b_j(\mathbf{o}_t))$$

This is also known as the Viterbi algorithm. It can be visualized as finding the best path through a matrix where the vertical dimension represents the states of the HMM and horizontal dimension represents frames of speech i.e. time. To complete the extension to connected speech recognition, it is further assumed that each HMM representing the underlying sequence is connected. Thus the training data for continuous speech recognition should consist of connected utterances; however, the boundaries between words do not have to be known.

To improve computational speed/efficiency, the Viterbi algorithm is sometimes extended to achieve convergence by using what is known as a Token Passing Model. The token passing model represents a partial match between the observation sequence $\mathbf{o}_1$ **to** $\mathbf{o}_t$ and a particular model, subject to the constraint that the model is in state j at time t. This token passing model can be extended easily to connected speech environments as well if we allow the sequence of HMMs to be defined as a finite state network. A composite network that includes both phoneme-based HMMs and complete words can be constructed so that a single-best word can be recognized to form connected speech using word N-best extraction from the lattice of possibilities. This composite form of HMM-based connected speech recognizer is the basis of the NLQS speech recognizer module. Nonetheless, the present invention is not limited as such to such specific forms of speech recognizers, and can employ other techniques for speech recognition if they are otherwise compatible with the present architecture and meet necessary performance criteria for accuracy and speed to provide a real-time dialog experience for users.

The representation of speech for the present invention's HMM-based speech recognition system assumes that speech is essentially either a quasi-periodic pulse train (for voiced speech sounds) or a random noise source (for unvoiced sounds). It may be modelled as two sources – one a impulse train generator with pitch period P and a random noise generator which is controlled by a voice/unvoiced switch. The output of the switch is then fed into a gain function estimated from the speech signal and scaled to feed a digital filter H(z) controlled by the vocal tract paarmeter characteristics of the speech being produced. All of the parameters for this model – the voiced/unvoiced switching, the pitch period for voiced sounds, the gain parameter for the speech signal and the coefficient of the digital filter, vary slowly with time. In extracting the acoustic parameters from the user's speech input so that it can evaluated in light of a set of HMMs, cepstral analysis is typically used to separate the vocal tract information from the excitation information.

The cepstrum of a signal is computed by taking the Fourier (or similar) transform of the log spectrum. The principal advantage of extracting cepstral coefficients is that they are de-correlated and the diagonal covariances can be used with HMMs. Since the human ear resolves frequencies non-linearly across the audio spectrum, it has been shown that a front-end that operates in a similar non-linear way improves speech recognition performance.

Accordingly, instead of a typical linear prediction-based analysis, the front-end of the NLQS speech recognition engine implements a simple, fast Fourier transform based filter bank designed to give approximately equal resolution on the Mel-scale. To implement this filter bank, a window of speech data is transformed using a software based Fourier transform and the magnitude taken. Each FFT magnitude is then multiplied by the corresponding filter gain and the results accumulated. The cepstral coefficients that are derived from this filter-bank analysis at the front end are calculated during a first partial processing phase of the speech signal by using a Discrete Cosine Transform of the log filter bank amplitudes. These cepstral coefficients are called Mel-Frequency Cepstral Coefficients (MFCC) and they represent some of the speech parameters transferred from the client side to characterize the acoustic features of the user's speech signal. These parameters are chosen for a number of reasons, including the fact that they can be quickly and consistently derived even across systems of disparate capabilities (i.e., for everything from a low power PDA to a high powered desktop system), they give good discrimination, they lend themselves to a number of useful recognition related manipulations, and they are relatively small and compact in size so that they can be transported rapidly across even a relatively narrow band link.

To augment the speech parameters an energy term in the form of the logarithm of the signal energy is added. Accordingly, RMS energy is added to the 12 MFCC's to make 13 coefficients. These coefficients together make up the partially processed speech data transmitted in compressed form from the user's client system to the remote server side.

The performance of the present speech recognition system is enhanced significantly by computing and adding time derivatives to the basic static MFCC parameters at the server side. These two other sets of coefficients -- the delta and acceleration coefficients representing change in each of the 13 values from frame to frame (actually measured across several frames), are computed during a second partial speech signal processing phase to complete the initial processing of the speech signal, and are added to the original set of coefficients after the latter are received. These MFCCs together with the delta and acceleration coefficients constitute the observation vector $\mathbf{O}_t$ mentioned above that is used for determining the appropriate HMM for the speech data.

The delta and acceleration coefficients are computed using the following regression formula:

$$d_t = \Sigma^{\theta}_{\theta=1} [c_{t+\theta} - c_{t-\theta}] / 2\Sigma^{\theta}_{\theta=1}\theta^2$$

where d_t is a delta coefficient at time t computed in terms of the corresponding static coefficients:

$$d_t = [c_{t+\theta} - c_{t-\theta}] / 2\theta$$

In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. In other words, both the first and second partial processing phases above are executed by the same DSP (or microprocessor) running a ROM or software code routine at the client's computing machine.

In contrast, because of several considerations, specifically - cost, technical performance, and client hardware uniformity, the present NLQS system uses a partitioned or distributed approach. While some processing occurs on the client side, the main speech recognition engine runs on a centrally located server or number of servers. More specifically, as noted earlier, capture of the speech signals, MFCC vector extraction and compression are implemented on the client's machine during a first partial processing phase. The primary MFCCs are then transmitted to the server over the channel, which, for example, can include a dial-up Internet connection, a LAN connection, a wireless connection and the like.

After decompression, the delta and acceleration coefficients are computed at the server to complete the initial speech processing phase, and the resulting observation vectors $\mathbf{O}_t$ are also determined.

General Aspects of Speech Recognition Engine

The speech recognition engine is also located on the server, and is based on a HTK-based recognition network compiled from a word-level network, a dictionary and a set of HMMs. The recognition network consists of a set of nodes connected by arcs. Each node is either a HMM model instance or a word end. Each model node is itself a network consisting of states connected by arcs. Thus when fully compiled, a speech recognition network consists of HMM states connected by transitions. For an unknown input utterance with T frames, every path from the start node to the exit node of the network passes through T HMM states. Each of these paths has log probability which is computed by summing the log probability of each individual transition in the path and th elog probability of each emitting state generating the corresponding observation. The function of

the Viterbi decoder is find those paths throuh the network which have the highest log probability. This is found uisng the Token Passing algorithm. In a network that has many nodes, the computation time is reduced by only allowing propagation of those tokens which will have some chance of becoming winners. This process is called pruning.

Natural Language Processor

In a typical natural language interface to a database, the user enters a question in his/her natural language, for example, English. The system parses it and translates it to a query language expression. The system then uses the query language expression to process the query and if the search is successful, a recordset representing the results is displayed in English either formatted as raw text or in a graphical form. For a natural langauge interface to work well involves a number of technical requirements.

For example, it needs to be robust – in the sentence '*What's the departments turnover*' it needs to decide that the word *whats = what's = what is*. And it also has to determine that *departments = department's*. In addition to being robust, the natural language interface has to distinguish between the several possible forms of ambiguity that may exist in the natural language – lexical, structural, reference and ellipsis ambiguity. All of these requirements in addition to the general ability to perform the basic linguistic morphological operations of tokenization, tagging and grouping are implemented within the present invention.

Tokenization is implemented by a text analyzer which treats the text as a series of tokens or useful meaningful units that are larger than individual characters, but smmaler than phrases and sentences. These include words, separable parts of words, and punctuation. Each token is associated with an offset and a length. The first phase of tokenization is the process of segmenttaion which extracts the individual tokens from the input text and keeps track of the offset where each token originated in the input text. The tokenizer ouput lists the offset and category for each token. In the next phase of the text analysis, the tagger uses a built-in morphological analyzer to look up each word/token in a phrase or sentence and interanlly lists all parts of speech. The output is the input string with each token tagged with a parts of speech notation. Finally the grouper which functions as a phrase extractor or phrase analyzer, determines which groups of words form phrases. These three operations which are the foundations for any modern linguistic processing schemes, are fully implemented in optimized algorithms for determining the single-best possible answer to the user's question.

[Note: I did not correct this next section to any large degree; you should verify that it is not inconsistent with the later detailed sections; I do not think it needs to be so long as an introduction … DONE …I deleted a lot of extraneous stuff and kept just what is reuired to explain the full-text querying description]

SQL Database and Full-Text Query

Another key component of present system is a SQL-database. This database is used to store text, specifically the the answer-question pairs are stored in full-text tables of the database. Additionally, the full-text search capability of the database allows full-text searches to be carried out.

While a large portion of all digitally stored information is in the form of unstructured data, primarily text, it is now ~~only~~ possible to store this textual data in traditional ~~relational~~ database ~~management~~ systems ~~(RDBMs)~~ in character-based columns such as **varchar** and **text**. In order to effectively retrieve textual data from the database, techniques have to be implemented to issue queries against textual data and to retrieve the answers in a meaningful way where it provides the answers as in the case of the NLQS system. ~~Traditional RDBMSs have not been designed for efficient full-text retrieval.~~ [***We will want to soften the "absoluteness" of this text to RDBMS systems …. DONE***]

There are two major types of textual searches: Property - This search technology first applies filters to documents in order to extract properties such as author, subject, type, word count, printed page count, and time last written, and then issues searches against those properties; Full-text –this search technology first creates indexes of all non-noise words in the documents, and then uses these indexes to support linguistic searches and proximity searches.

~~The following~~ ~~t~~Two additional technologies are also implemented in this particular RDBMs:SQL Server also have been integrated: A Search service - a full-text indexing and search service that is called both index engine and search, and a parser that accepts full-text SQL extensions and maps them into a form that can be processed by the search engine.~~engine in the context of this document. Within the context of SQL Server, this is called full-text search; the parser component of the OLE DB Provider for Index Server 2.0 that accepts full-text SQL extensions and maps them into a form that can be processed by the search engine.~~

The four major aspects involved in implementing full-text retrieval of plain-text data from a full-text-capable ~~RDBMs~~ databases are: Managing the definition of the tables and columns that are registered for full-text searches; Indexing the data in registered columns - the indexing process scans

the character streams, determines the word boundaries (this is called word breaking), removes all noise words (this also is called stop words), and then populates a full-text index with the remaining words; Issuing queries against registered columns for populated full-text indexes; Ensuring that subsequent changes to the data in registered columns gets propagated to the index engine to keep the full-text indexes synchronized.

The underlying design principle for the indexing, querying, and synchronizing processes is the presence of a full-text unique key column (or single-column primary key) on all tables registered for full-text searches. The full-text index contains an entry for the non-noise words in each row together with the value of the key column for each row.

When processing a full-text search, the search engine returns to the ~~RDBMs~~ database the key values of the rows that match the search criteria.

~~Unlike classic database indexes, traditional full-text indexes are not modified instantly when values in full-text registered columns are updated, when rows are added to full-text registered tables, or when rows are deleted from full-text registered tables. Rather, full-text indexes usually are re-populated asynchronously. There are two reasons for this:~~

- ~~It typically takes significantly more time to update a full-text index than a classic index.~~
- ~~Full-text searches usually are fuzzy by nature and so do not need to be as precise as classic searches.~~

~~During repopulation, the unique key column values are passed to the index engine to identify those items that need to be re-indexed. For example, if the title associated with V109 gets changed to "Mystery Island," then the index should be modified to reflect this new value.~~

The full-text administration process starts by designating a table and its columns of interest for full-text search. Customized NLQS stored procedures are used first to register tables and columns as eligible for full-text search. After that, a separate request by means of a stored procedure is issued to populate the full-text indexes. ~~[see other patent]~~

The result is that the underlying index engine gets invoked and asynchronous index population begins. Full-text indexing tracks which significant words are used and where they are located. For example, a full-text index might indicate that the word "NLQS" is found at word number 423 and word number 982 in the **Abstract** column of the **DevTools** table for the row associated with a **ProductID** of 6. This index structure supports an efficient search for all items containing indexed words as well as advanced search operations, such as phrase searches and

proximity searches. (An example of a phrase search is looking for "*white elephant*," where "*white*" is followed by "*elephant*". An example of a proximity search is looking for "*big*" and "*house*" where "*big*" occurs near "*house*".) To prevent the full-text index from becoming bloated, noise words such as "*a*," "*and*," and "*the*" are ignored.

Extensions to the Transact-SQL language are used to construct full-text queries. The two key predicates that are used in the NLQS are **CONTAINS** and **FREETEXT.**

The **CONTAINS** predicate is used to determine whether or not values in full-text registered columns contain certain words and phrases. Specifically, this predicate is used to search for:

- A word or phrase.
- The prefix of a word or phrase.
- A word or phrase that is near another.
- A word that is an inflectional form of another (for example, "*drive*" is the inflectional stem of "*drives*," "*drove*," "*driving*," and "*driven*").
- A set of words or phrases, each of which is assigned a different weighting.

~~The **FREETEXT** predicate is a basic form of a natural language query. It is used to determine whether or not values in full-text registered columns reflect the meaning, rather than the exact words, specified in the predicate. Like a natural language processor, the index engine of RDBMS breaks the free text string into a number of search terms, generates the stemmed form of the words, assigns heuristic weighting to each term, and then finds the matches. Any text, including words, phrases, or sentences, can be specified in the query. The RDBMS search engine matches values that reflect the meaning, rather than the exact wording, of the query.~~

The relational engine within SQL Server recognizes the **CONTAINS** and **FREETEXT** predicates and performs some minimal syntax and semantic checking, such as ensuring that the column referenced in the predicate has been registered for full-text searches. During query execution, a full-text predicate and other relevant information are passed to the full-text search component. After further syntax and semantic validation, the search engine is invoked and returns the set of unique key values identifying those rows in the table that satisfy the full-text search condition. In addition to the **FREETEXT** and **CONTAINS**, other predicates such as **AND**, **LIKE**, **NEAR** are combined to create the customized NLQS SQL construct.

Full-Text Query Architecture of the SQL Database

The full-text query architecture is comprised of the following several components – Full-Text Query component, the SQL Server Relational Engine, the Full-Text provider and the Search Engine.

The **Full-Text Query** component of the SQL database accept a full-text predicate or rowset-valued function from the SQL Server; transform parts of the predicate into an internal format, and sends it to ~~RDBMS~~ Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the resultset that is then returned to the submitter of the query.

The **SQL Server Relational Engine** accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column that has not been registered for full-text search. If valid, then at run time, the ft_search_condition and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query. The **Full-Text Provider** parses and validates the ft_search_condition, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy ft_search_condition. ~~The handling of this rowset is conceptually similar to the code used in support of the **OPENROWSET()** and **OPENQUERY()** rowset-valued functions.~~

[At this point, the Client Side System 150 and Server Side System 180 will be explained in detail]

Although the present invention has been described in terms of a preferred embodiment, it will be apparent to those skilled in the art that many alterations and modifications may be made to such embodiments without departing from the teachings of the present invention. It will also be apparent to those skilled in the art that for purposes of the present discussion, the block diagram of the present invention has been simplified. The microcode and software routines executed to effectuate the inventive methods may be embodied in various forms, including in a permanent magnetic media, a non-volatile ROM, a CD-ROM, or any other suitable machine-readable format. Accordingly, it is intended that the all such alterations and modifications be included within the scope and spirit of the invention as defined by the following claims.

What is claimed is:

[These will be filled in on an application by application basis]

ABSTRACT OF THE DISCLOSURE

Since we have 4 applications, 4 different abstracts should be written.

Distributed Real Time Speech Recognition System

A real-time system incorporating speech recognition and linguistic processing for recognizing a spoken query by a user and distributed between client and server, is disclosed. The system accepts user's queries in the form of speech at the client where minimal processing extracts a sufficient number of acoustic speech vectors representing the utterance. These vectors are sent via a communications channel to the server where additional acoustic vectors are derived. Using Hidden Markov Models (HMMs), and appropriate grammars and dictionaries conditioned by the selections made by the user, the speech representing the user's query is fully decoding to text at the server. This text corresponding to the user's query is then simultaneously sent to a natural language engine and a database processor where optimized SQL statements are constructed for a full-text search from a database for a recordset of several stored questions that best matches the user's query. Further processing in the natural language engine narrows the search to a single stored question. The answer corresponding to this single stored question is next retrieved from the file path and sent to the client in compressed form. At the client, the answer to the user's query is articulated to the user using a text-to-speech engine in his or her native natural language. The system requires no training and can operate in several natural languages.

Speech Based Learning/Training System

A real-time speech-based learning /training system distributed between client and server, and incorporating speech recognition and linguistic processing for recognizing a spoken question and to provide an answer to the student in a learning or training environment implemented on an intranet or over the Internet, is disclosed. The system accepts the student's question in the form of speech at his or her computer, PDA or workstation where minimal processing extracts a sufficient number of acoustic speech vectors representing the utterance. The system as implemented accepts environmental variables such as course, chapter, section as selected by the user so that the search time, accuracy and response time for the question can be optimized. A minimum set of acoustic vectors extracted at the client are then sent via a communications channel to the server where additional acoustic vectors are derived. Using Hidden Markov Models (HMMs), and appropriate grammars and dictionaries conditioned by the course, chapter and section selections made by the student, the speech representing the user's query is fully decoding to text at the server. This text corresponding to the user's query is then simultaneously sent to a natural language engine and a database processor where an optimized SQL statement is constructed for a full-text search from a SQL database for a recordset of several stored questions that best matches the user's query. Further processing in the natural language engine narrows the search down to a single stored question. The answer that is paired to this single stored question is then retrieved from the file path and sent to the student computer in compressed form. At the student's computer, the answer is articulated using a text-to-speech engine in his or her native natural language. The system requires no training and can operate in several natural languages.

Internet Server with Speech Support for Enhanced Interactivity

An Internet-based server with speech support for enhanced interactivity is disclosed. This server hosts a server-side speech recognition engine and additional linguistic and database functions that cooperate to provide enhanced interactivity for clients so that their browsing experience is more satisfying, efficient and productive. This human-like interactivity which allows the user to ask queries about topics that range from customer delivery, product descriptions, payment details, is facilitated by the allowing the user to articulate the his or her questions directly in his or her natural language. The answer typically provided in real-time, can also be interfaced and integrated with existing telephone, e-mail and other mixed media services to provide a single point of interactivity for the user when browsing at a web-site.

Intelligent Query System for Processing Voice Based Queries

An intelligent query system for processing voiced-based queries is disclosed. This distributed client-server system, typically implemented on an intranet or over the Internet accepts a user's queries at his/her computer, PDA or workstation using a speech input interface. After converting the user's query from speech to text, a 2-step algorithm employing a natural language engine, a database processor and a full-text SQL database is implemented to find a single answer that best matches the user's query. The first step of the 2-step algorithm formulates the optimized SQL construct for searching a SQL full-text database that contains similar stored questions as the user's question. The SQL construct includes a metric derived from a natural language morphological analysis of the user's query, in addition to the full-text predicates required to perform the full-text search. The morphological analysis carried out within the natural language engine includes tokenization, stemming and grouping of the user's query so that the metric derived – noun phrase frequency can be applied as a parameter in each of the 2 steps. Once the search results are returned by the SQL full-text search, the second step of the 2-step algorithm implemented by the natural language engine is initiated to narrow the search from a set of possible paired questions to a single question that best matches the user's query. The answer that is paired to this single question is then retrieved and presented to the user. The system as implemented, accepts environmental variables selected by the user and is scalable to provide answers to a variety and quantity of user-initiated queries.

Client System 150

The architecture of client-side system 150 of Natural Language Query System 100 is illustrated in greater detail in **FIG. 2**. Referring to **FIG. 2**, the three main processes effectuated by Client System 150 are illustrated as follows: Initialization process **200A** consisting of SRE **201,** Communication **202** and MS Agent **203 routines;** an iterative process **200B** consisting of two sub-routines: a) Receive User Speech **208** – made up of SRE **204 and Communication 205;** and b) Receive Answer from Server **207** and an un-initialization process **200B**. Finally, un-initialization process 200C is made up of three sub-routines: SRE **212,** Communication **213**, and MS Agent **214.** Each of the above three processes are described in detail in the following paragraphs. It will be appreciated by those skilled in the art that the particular implementation for such processes and routines will vary from client platform to platform, so that in some environments such processes may be embodied in hard-coded routines executed by a dedicated DSP, while in others they may be embodied as software routines executed by a shared host processor, and in still others a combination of the two may be used.

Deleted: ¶

Deleted: Fig

Deleted: Fig

Deleted: processes

Deleted: processes

Deleted: The

Deleted: processes

Deleted: ¶
¶

Deleted:

Initialization at Client System 150

The initialization of the Client System 150 is illustrated in **FIG. 2-2** and is comprised generally of 3 separate initializing processes: client-side Speech Recognition Engine **220A,** MS Agent **220B** and Communication processes **220C**.

Initialization of Speech Recognition Engine 220A

Speech Recognition Engine 155 is initialized and configured using the routines shown in 220A. First, an SRE COM Library is initialized. Next, memory **220** is allocated to hold Source and Coder objects, which represent ………………………………. And which are created by a routine 221. Loading of configuration file **221A** from configuration data file **221B** also takes place at the same time that the SRE Library is initialized. In configuration file **221B**, the type of the input of Coder and the type of the output of the Coder are declared. The structure, operation, etc. of such routines are well-known in the art, and they can be implemented using a number of fairly straightforward approaches. Accordingly, they are not discussed in detail herein. Next, Speech and Silence components of an utterance are

Deleted: ¶

Deleted: client side

Deleted: as

Deleted: Fig

Deleted: 3

Deleted: the

Deleted: the

Deleted: the

Deleted: **the**

Deleted: ¶

Deleted: In block **1** the SRE COM Library is initialized, and

Deleted: the

Deleted: objects of Source and Coder. Source and Coder objects **221**are created

Deleted: the

Deleted: the

Deleted: ,

Deleted: ¶
¶

Deleted: the

Revised: 9/13/99

calibrated using a routine **222**, in a procedure that is also well-known in the art. To calibrate the speech and silence components, the user preferably articulates a sentence that is displayed in a text box on the screen. The SRE library then estimates the noise and other parameters required to find e silence and speech elements of future user utterances.

Deleted: .
Deleted: Then the
Deleted: th
Deleted: the
Deleted: ¶
Deleted: the
Deleted: ¶
Deleted: Initialization
Deleted: described by block **220B** as
Deleted: Fig
Deleted: the

Initialization of MS Agent 220B

The software code used to initialize and set up a MS Agent 220B is also illustrated in **FIG. 2-2.** The MS Agent 220B routine is responsible for coordinating and handling the actions of the animated agent 157 (FIG.1). This initialization thus consists of the following steps:

1. Initialize COM library **223**. This part of the code initializes the COM library, which is required to use ActiveX Controls, which controls are well-known in the art.
2. Create instance of Agent Server **224** - this part of the code creates an instance of Agent ActiveX control.
3. Loading of MS Agent **225** - this part of the code loads MS Agent character from a specified file **225A** containing general parameter data for the Agent Character, such as the overall appearance, shape, size, etc..
4. Get Character Interface **226** - this part of the code gets an appropriate interface for the specified character; for example, characters may have different control/interaction capabilities that can be presented to the user.
5. Add Commands to Agent Character Option **227** - this part of the code adds commands to an Agent Properties sheet, which sheet can be accessed by clicking on the icon that appears in the system tray, when the Agent character is loaded *e.g.* that the character can Speak, how he/she moves, TTS Properties, etc.
6. Show the Agent Character **228** - this part of the code displays the Agent character on the screen so it can be seen by the user;
7. AgentNotifySink - to handle events. This part of the code creates AgentNotifySink object **229**, registers it at **230** and then gets the Agent Properties interface **231**. The property sheet for the Agent character is assigned using routine **232**.
8. Do Character Animations **233** - This part of the code plays specified character animations to welcome the user to NLQS 100.

Deleted: the
Deleted: the
Formatted
Deleted: the
Formatted
Deleted: .
Deleted: Get
Deleted: to
Deleted: done in
Deleted: the

The above then constitutes the entire sequence required to initialize the MS Agent. As with the SRE routines, the MS Agent routines can be implemented in any suitable and conventional fashion by those skilled in the art based on the present teachings. The

Revised: 9/13/99

particular structure, operation, etc. of such routines is not critical, and thus they are not discussed in detail herein.

In a preferred embodiment, the MS Agent is configured to have an appearance and capabilities that are appropriate for the particular application. For instance, in a remote learning application, the agent has the visual form and mannerisms/attitude/gestures of a college professor. Other visual props (blackboard, textbook, etc.) may be used by the agent and presented to the user to bring to mind the experience of being in an actual educational environment. The characteristics of the agent may be configured at the client side 150, and/or as part of code executed by a browser program (not shown) in response to configuration data and commands from a particular web page. For example, a particular website offering medical services may prefer to use a visual image of a doctor. These and many other variations will be apparent to those skilled in the art for enhancing the human-like, real-time dialog experience for users.

Initialization of Communication Link 160A

The initialization of Communication Link 160A is shown with reference to process **220C FIG. 2-2**. Referring to **FIG. 2-2**, this initialization consists of the following code components: Open Internet Connection **234** - this part of the code opens an Internet Connection and sets the parameter for the connection. Then Set Callback Status routine **235** sets the callback status so as to inform the user of the status of connection. Finally Start New HTTP Internet Session **236** starts a new Internet session. The details of Communications Link 160 and the set up process 220C are not critical, and will vary from platform to platform. Again, in some cases, users may use a low-speed dial-up connection, a dedicated high speed switched connection (T1 for example), an always-on xDSL connection, a wireless connection, and the like.

Iterative Processing of Queries/Answers

As illustrated in **FIG. 3**, once initialization is complete, an iterative query/answer process is launched when the user presses the Start Button to initiate a query. Referring to **FIG. 3**, the iterative query/answer process consists of two main sub-processes implemented as routines on the client side system 150: **Receive User Speech 240** and **Receive User Answer 243.** The **Receive User Speech 240** routine receives speech from the user (or

Deleted: ¶
¶

Deleted: the

Deleted: Process

Deleted: the

Deleted: P

Deleted: is also illustrated in

Deleted: Fig

Deleted: Fig

Deleted: the

Deleted: Fig

Deleted: the

Deleted: Fig

Deleted: the

Revised: 9/13/99

another audio input source), while the **Receive User Answer 243** routine receives an answer to the user's question in the form of text from the server so that it can be converted to speech for the user by text-to-speech engine 159. As used herein, the term "query" is referred to in the broadest sense to refer, to either a question, a command, or some form of control request to the server. For example, a query may consist of a question directed to a particular topic, such as "what is a network" in the context of a remote learning application. In an e-commerce application a query might consist of a command to "list all books by Mark Twain" for example. Similarly, while the answer in a remote learning application consists of text that is rendered into audible form by the text to speech engine 159, it could also be returned as another form of multi-media information, such as a graphic image, a sound file, a video file, etc. depending on the requirements of the particular application.

Again, given the present teachings concerning the necessary structure, operation, functions, performance, etc., of the client-side Receive User Speech 240 and Receiver User Answer 243 routines, one of ordinary skill in the art could implement such in a variety of ways.

Receive User Speech – As illustrated in **FIG. 3**, the Receive User Speech routine 240 consists of a SRE **241** and a Communication **242** process, both implemented again as routines on the client side system 150 for receiving and partially processing the user's utterance. SRE routine 241 uses a coder **248** which is prepared so that a coder object receives speech data from a source object. Next the Start Source **249** routine is initiated. This part of the code initiates data retrieval using the source Object which will in turn be given to the Coder object. Next, MFCC vectors **250** are extracted from the Speech utterance continuously until silence is detected. As alluded to earlier, this represents the first phase of processing of the input speech signal, and in a preferred embodiment, it is intentionally restricted to merely computing the MFCC vectors for the reasons already expressed above. These vectors include the 12 cepstral coefficients and the RMS energy term, for a total of 13 separate numerical values for the partially processed speech signal.

In some environments, nonetheless, it is conceivable that the MFCC delta parameters and MFCC acceleration parameters can also be computed at client side system 150, depending on the computation resources available, the transmission bandwidth in data link 160A available to server side system 180, the speed of a transceiver used for carrying

Deleted: the
Deleted: the
Deleted: .
Deleted: Fig
Deleted: the
Deleted: the
Deleted: es are the two (2) processes involved in
Deleted: The
Deleted: the
Deleted: the
Formatted
Deleted: the
Deleted:

Revised: 9/13/99

data in the data link, etc. These parameters can be determined automatically by client side system upon initializing SRE 155 (using some type of calibration routine to measure resources), or by direct user control, so that the partitioning of signal processing responsibilities can be optimized on a case-by-case basis. In some applications, too, server side system 180 may lack the appropriate resources or routines for completing the processing of the speech input signal. Therefore, for some applications, the allocation of signal processing responsibilities may be partitioned differently, to the point where in fact both phases of the speech signal processing may take place at client side system 150 so that the speech signal is completely - rather than partially –processed and transmitted for conversion into a query at server side system 180.

Again in a preferred embodiment, to ensure reasonable accuracy and real-time performance from a query/response perspective, sufficient resources are made available in a client side system so that 100 frames per second of speech data can be partially processed and transmitted through link 160A. Since the least amount of information that is necessary to complete the speech recognition process (only 13 coefficients) is sent, the system achieves a real-time performance that is believed to be highly optimized, because other latencies (i.e., client-side computational latencies, packet formation latencies, transmission latencies) are ~~eliminated~~ minimized. It will be apparent that the principles of the present invention can be extended to other SR applications where some other methodology is used for breaking down the speech input signal by an SRE (i.e., non-MFCC based). The only criteria is that the SR processing be similarly dividable into multiple phases, and with the responsibility for different phases being handled on opposite sides of link 160A depending on overall system performance goals, requirements and the like.

Thus, the present invention achieves a response rate performance that is tailored in accordance with the amount of information that is computed, coded and transmitted by the client side system 150. So in applications where real-time performance is most critical, the least possible amount of extracted speech data is transmitted to reduce these latencies, and, in other applications, the amount of extracted speech data that is processed, coded and transmitted can be varied.

Communication – transmit communication module **242** is used to implement the transport of data from the client to the server over the data link 160A, which in a preferred embodiment is the Internet. As explained above, the data consists of encoded MFCC

Deleted:

Deleted:

Deleted:

Formatted

Deleted:

Deleted: ¶
¶

Deleted: the

Deleted: T

Revised: 9/13/99

vectors that will be used at then server-side of the Speech Recognition engine to complete the speech recognition decoding. The sequence of the communication is as follows:

OpenHTTPRequest 251 - this part of the code first converts MFCC vectors to a stream of bytes, and then processes the bytes so that it is compatible with a protocol known as HTTP. This protocol is well-known in the art, and it is apparent that for other data links another suitable protocol would be used.

1. **Encode MFCC Byte Stream 251** - this part of the code encodes the MFCC vectors, so that they can be sent to the server via HTTP.
2. **Send data 252** - this part of the code sends MFCC vectors to the server using the Internet connection and the HTTP protocol.

Wait for the Server Response 253 - this part of the code monitors the data link 160A a response from server side system 180 arrives. In summary, the MFCC parameters are extracted or observed on-the-fly from the input speech signal. They are then encoded to a HTTP byte stream and sent in a streaming fashion to the server before the silence is detected – i.e. sent to server side system 180 before the utterance is complete. This aspect of the invention also facilitates a real-time behavior, since data can be transmitted and processed even while the user is still speaking.

Receive Answer from Server 243 is comprised of the following modules as shown in **FIG. 3**.: MS Agent **244**, Text-to-Speech Engine **245** and receive communication modules **246**. All three modules interact to receive the answer from server side system 180. As illustrated in **FIG. 3**, the receive communication process consists of three separate processes implemented as a receive routine on client side system 150: a Receive the Best Answer **258** receives the best answer over data link 160B (the HTTP communication channel). The answer is de-compressed at **259** and then the answer is passed by code 260 to the MS Agent 244, where it is received by code portion 254. A routine 255 then articulates the answer using text-to-speech engine **257**. Of course, the text can also be displayed for additional feedback purposes on a monitor used with client side system 150. The text to speech engine uses a natural language voice data file **256** associated with it that is appropriate for the particular language application (i.e., English, French, German, Japanese, etc.). As explained

Deleted: the HTTP protocol. ¶
Formatted: Bullets and Numbering
Deleted: the
Deleted: it
Deleted: http
Deleted: (Regarding the encoding more information is given in the server side document)
Deleted: the
Deleted: http
Deleted: sends will be in
Deleted: loop till
Deleted: the
Deleted: the
Deleted: ¶
Formatted
Deleted:
Deleted: Fig
Deleted: the
Deleted: the
Deleted: C
Deleted: the
Deleted: Fig
Deleted: :
Deleted: the
Deleted: in
Deleted: by 260
Formatted
Deleted: MS Agent 254 receives the answer from Communication process 246
Deleted: MS Agent 255
Deleted: the
Deleted: the
Deleted: that is
Formatted

Revised: 9/13/99

earlier when the answer is something more than text, it can be treated as desired to provide responsive information to the user, such as with a graphics image, a sound, a video clip, etc.

Uninitialization

The un-initialization routines and processes are illustrated in FIG. **4**. Three functional modules are used for un-initializing the primary components of the client side system 150; these include SRE **270**, Communications **271** and MS Agent **272** un-initializing routines. To un-initialize SRE 220A, memory that was allocated in the initialization phase is de-allocated by code **273** and objects created during such initialization phase are deleted by code **274.** Similarly, as illustrated in FIG. **4**, to un-initialize Communications module 220C the Internet connection previously established with the server is closed by code portion **275** of the Communication Un-itialization routine 271. Next the Internet session created at the time of initialization is also closed by routine **276.** For the un-initialization of the MS Agent 220B, as illustrated in FIG. **4**, MS Agent Un-initialization routine 272 first releases the Commands Interface 227 using routine **277**. This releases the commands added to the property sheet during loading of the agent character by routine 225. Next the Character Interface initialized by routine 226 is released by routine **278** and the Agent is unloaded at **279**. The Sink Object Interface is then also released **280** followed by the release of the Property Sheet Interface **281**. The Agent Notify Sink **282** then un-registers the Agent and finally the Agent Interface **283** is released which releases all the resources allocated during initialization steps identified in FIG. 2-2.

It will be appreciated by those skilled in the art that the particular implementation for such un-initialization processes and routines in FIG. 4 will vary from client platform to client platform, so that in some environments such processes may be embodied in hard-coded routines executed by a dedicated DSP, while in others they may be embodied as software routines executed by a shared host processor, and in still others a combination of the two may be used. The structure, operation, etc. of such routines are well-known in the art, and they can be implemented using a number of fairly straightforward approaches. Accordingly, they are not discussed in detail herein.

Deleted: is
Deleted: Fig
Deleted: s
Deleted: un-initialized – the
Deleted: ,
Deleted: the
Formatted
Deleted: the
Formatted
Formatted
Deleted: A
Deleted: Fig
Deleted: the
Deleted: 271, and
Formatted
Formatted
Deleted: d
Deleted: 272
Deleted: Fig
Deleted: is first released
Deleted: is
Deleted: ed
Deleted: .
Deleted:

Revised: 9/13/99

EXHIBIT 2

LAW OFFICES
KEKER & VAN NEST
LLP
710 SANSOME STREET
SAN FRANCISCO, CA 94111-1704
TELEPHONE (415) 391-5400
FAX (415) 397-7188
WWW.KVN.COM

SONALI D. MAITRA
SMAITRA@KVN.COM

August 28, 2008

VIA FACSIMILE AND U.S. MAIL

R. Joseph Trojan, Esq.
Trojan Law Offices
9250 Wilshire Boulevard, Suite 325
Beverly Hills, CA 90212

RE: *Phoenix Solutions, Inc. v. Wells Fargo Bank, N.A.*
No. CV-08-0863 MHP

Dear Joe:

Thank you for taking the time to speak with us yesterday afternoon regarding the scope of Phoenix's subject matter waiver of the attorney-client privilege ("the waiver"). I am writing to confirm the results of our meet-and-confer yesterday. Please let me know immediately if you disagree with any of the below.

We understand from our discussion that Phoenix carefully reviewed all of the more than 10,000 pages it produced as part of its initial disclosures and, as part of that process, intentionally produced all of the communications in Mr. Gross's files relating to the preparation of the patent applications that pre-date the filing of the patent applications. You stated that it was therefore your belief that all privileged attorney-client communications relating to the preparation of the patent applications that pre-date the filing of the patent applications were already produced. You were uncertain whether that meant that all communications between Mr. Bennett and Mr. Gross prior to the date of filing the patent applications had been produced, but indicated that you would check with Mr. Gross to determine whether that was the case.

Turning to the consequences of Phoenix's decision to produce those privileged documents, the parties agreed that the scope of the waiver extends to: (1) communications relating to editing and drafting of the specification of the patents, including the scope and content of the prior art; (2) communications relating to the inventorship of the patents; and (3) communications relating to the novelty and/or innovation of the patents. The parties disagree on whether the waiver extends to communications relating to how to uncover prior art; Wells Fargo believes that it does, and Phoenix contends that it does not. And Phoenix is still considering whether the waiver extends to: (1) communications relating to how prior art applies to the invention; and (2) communications relating to how to draft the claims to avoid prior art.

R. Joseph Trojan, Esq.
August 28, 2008
Page 2

The parties disagree on the temporal scope of the subject matter waiver. Phoenix contends that the waiver extends only through the date of filing of the patent applications with the Patent and Trademark Office; Wells Fargo does not agree that the waiver is so limited. Phoenix provided two cases in support of its position, which Wells Fargo will review and consider before a further meet-and-confer. We will contact you next week to continue this discussion in the hopes that the parties can reach agreement on the question of the scope of the waiver.

Thank you for your attention to this matter.

Sincerely,

Sonali D. Maitra

EXHIBIT 3

R. JOSEPH TROJAN

DYLAN C. DANG
JEEYEON HAN
SHARON E. GHAUSI
WILLIAM WONG
LLOYD VU

OF COUNSEL
J. NICHOLAS GROSS

TROJAN LAW OFFICES
Rexford Plaza
9250 Wilshire Boulevard
Suite 325
Beverly Hills, California 90212
www.patenTrademark.com
REGISTERED PATENT ATTORNEYS

PATENT, TRADEMARK,
COPYRIGHT, TRADE SECRET &
RELATED CAUSES

TELEPHONE (310) 777-8399
FACSIMILE (310) 777-8348

September 3, 2008

Sonali D. Maitra, Esq.
Keker & Van Nest, LLP
710 Sansome Street
San Francisco, California 94111-1704

VIA EMAIL ONLY

Re: [Phoenix]
Phoenix Solutions, Inc. v. Wells Fargo Bank, N.A.
U.S.D.C., Northern District of California
Case No. CV08-0863 MHP
TLO File No. 08-02-4824

Dear Ms. Sonali D. Maitra:

I am writing in response to your letter dated August 28, 2008 in which you discussed our meet and confer regarding the scope of Phoenix's waiver of the attorney-client privilege.

According to the letter, Wells Fargo believes that "Phoenix carefully reviewed all of the more than 10,000 pages it produced as part of its initial disclosures and, as part of that process, intentionally produced all of the communications in Mr. Gross's files relating to the preparation of the patent applications that pre-date the filing of the patent applications." That statement misstates Phoenix's position. A more accurate statement of Phoenix's position is that Phoenix believes that, after a comprehensive search through pre-filing communications between J. Nicholas Gross and Dr. Ian Bennett, Phoenix has produced all pre-filing communications between Mr. Gross and Dr. Bennett relating to the drafting of the specification of the patents-in-suit. In other words, Phoenix had no intention of selectively producing attorney-client communications, but rather, Phoenix believed that it produced all pre-filing communications between Mr. Gross and Dr. Bennett relating to the drafting of the specification of the patents-in-suit.

Also, the parties agreed that the scope of the waiver extends to (1) communications relating to the editing and drafting of the patents, including the scope of and content of the prior art *to the extent it relates to the drafting of the specification, i.e., to the extent that there is a duty to disclose prior art*; (2) communications relating to the inventorship of the patents *to the extent it relates to the drafting of the specification*; and (3) communications relating to the novelty and/or innovation of the patents *to the extent it relates to the drafting of the specification.* We

were adamant during the meet and confer that the scope of waiver was couched within the drafting of the specification.

If you have any further questions, feel free to contact us.

Very truly yours,

TROJAN LAW OFFICES
by

/s/ R. Joseph Trojan
R. Joseph Trojan

RJT:ww